Justice Information Network Data Exchange (JINDEX)

Traffic Records Project
Logical Design Document

Washington Department of
Information Services (DIS)

1110 Jefferson Street SE
PO Box 42445
Olympia, Washington 98504-2445

Revision & Sign-off Sheet

Change Record

Date	Author	Version	Change Reference
05/11/06	Rick Jenness	1.8	Phase 1 Final version
09/22/06	Rick Jenness	2.0	Phase 2 Draft Version updated with new WSDL specification
10/26/06	Rick Jenness	2.1	Phase 2 Final version

Reviewers

Name	Version Approved	Position	Date
Scott Bream			

Document Properties

Item	Details
Document Name	E-Trip Logical Design - Final.doc
Author	Andrei Kossoroukov & Rick Jenness
Creation Date	January 23, 2006
Last Updated	1/12/2007 7:58 AM, by Rick Jenness

Deleted: 10/27/2006 5:07 PM

Table of Contents

Deleted: 18

Deleted: 33

Deleted: 36

Deleted: 58

Table of Tables

1.0 E-Trip System

1.1 E-Trip System Overview

The Justice Information Network Data Exchange (JINDEX) is a message brokering service created by the Washington Integrated Justice Information Board (WIJIB) that provides the means by which Justice related agencies in the State share key information and business processes. The WIJIB has established that the JINDEX will be the foundation for justice information sharing projects within the State enterprise and will be designed to serve the diverse justice requirements of State & Local government entities as well as appropriate Federal and quasi-governmental entities operating in the State of Washington.

Based on the Microsoft BizTalk platform, and deployed on servers hosted at the Washington Department of Information Services (DIS), JINDEX is a Service Oriented Architecture (SOA) platform that will use Web Services and Justice XML to give participants the ability to exchange information and conduct transactions reliably, in real time, consistent with the individual operational requirements of its stakeholders.

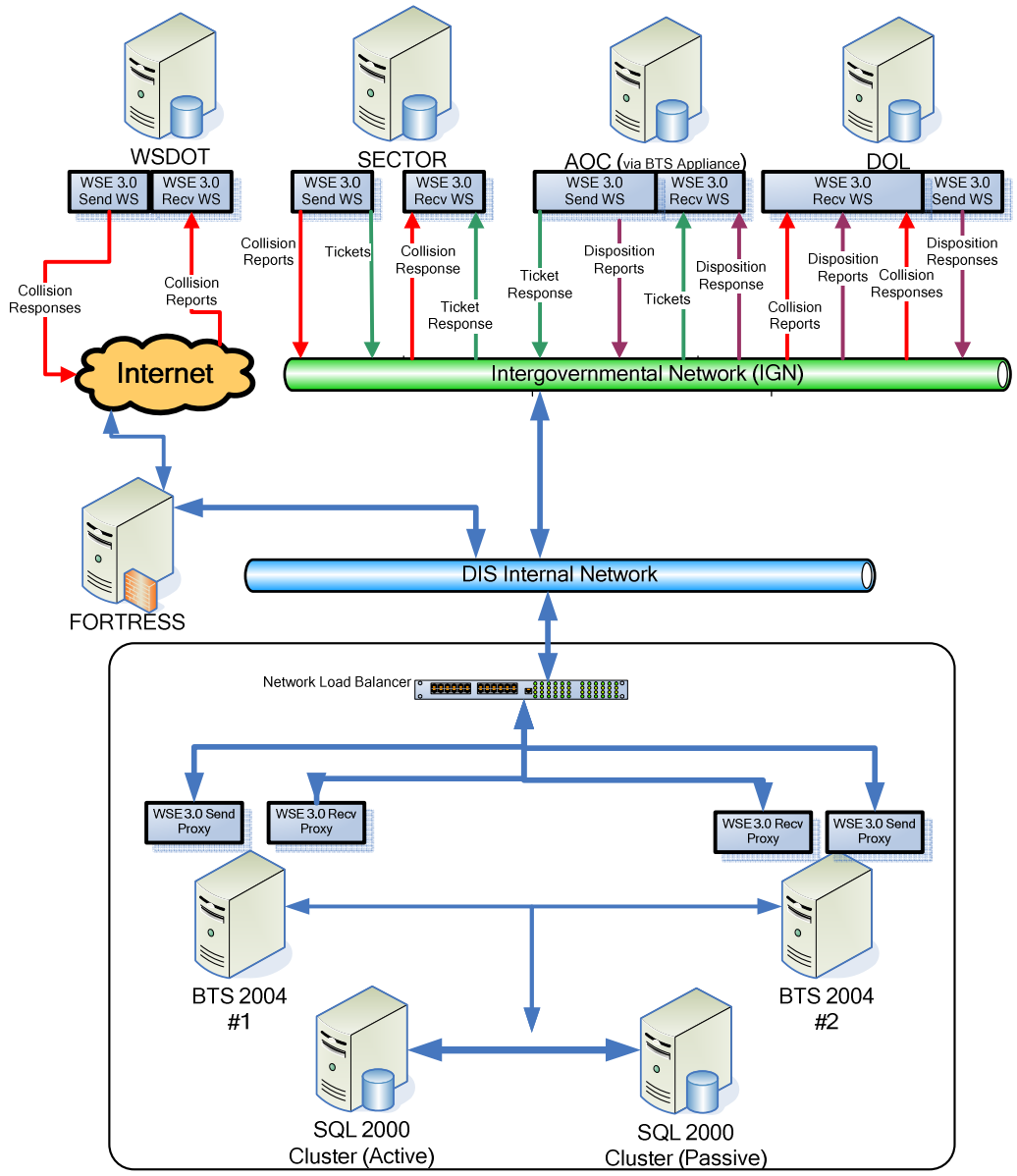
Each year, over one million tickets[1] are written for traffic and vehicle infractions in the State of Washington. Additionally, nearly 150,000 vehicle collisions reports and other forms are created by State and Local Law Enforcement officers as a part of their patrol assignments. Currently, all of these forms are created by hand, and entered into various computer systems around the state, some as many as four separate times. This manual process is prone to errors, time consuming, and very costly to taxpayers.

Stakeholders in this process have long envisioned a process where this data can be entered once and automatically be processed from the officer's patrol car, to their local or regional processing center, through the state court system and eventually archived in one of several state data repositories, all without ever having to re-enter the data. The end goal of this initiative is to speed processing, eliminate data entry errors, and minimize the manual effort processing information gathering forms by:

1. Automating data transfers between various information consuming agencies, and
2. Providing timely processing of pertinent and permitted information between data aggregating and consuming agencies.

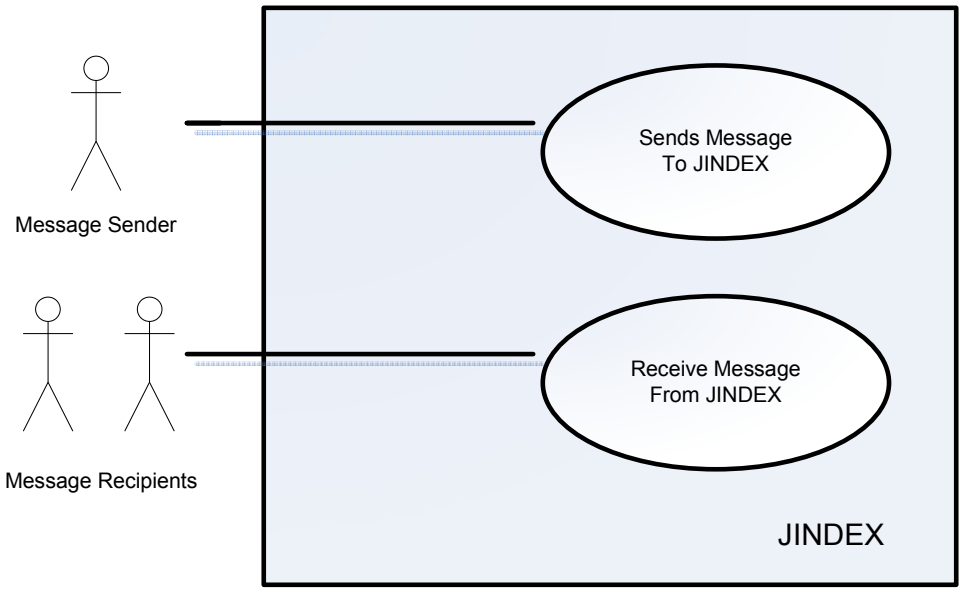
[1] Discussions surrounding the E-TRIP project included numerous references to a "Citation", "Ticket", and "Notice of Infraction" somewhat interchangeably. Accurately stated, "Citation" is a term used by the Washington Rules of Court to describe a charging document for criminal cases and a "Notice of Infraction" is the document used for non-criminal infractions. In using the generic term "Ticket" in this document,we are referring to both.

1.2 General System Architecture

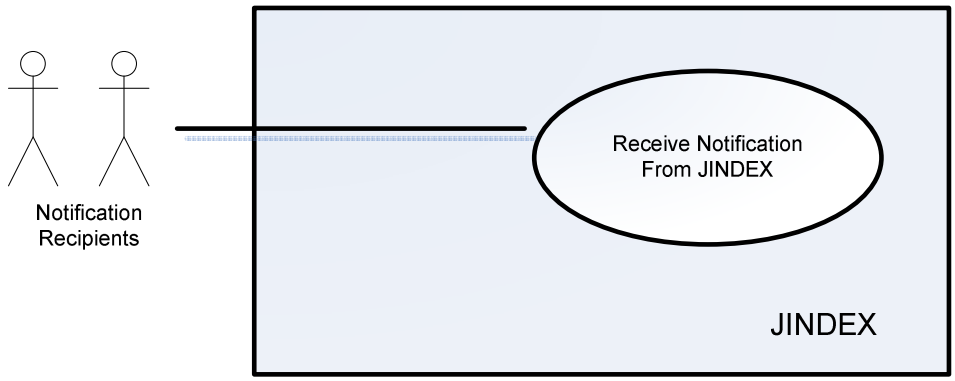


1.3 Use Cases

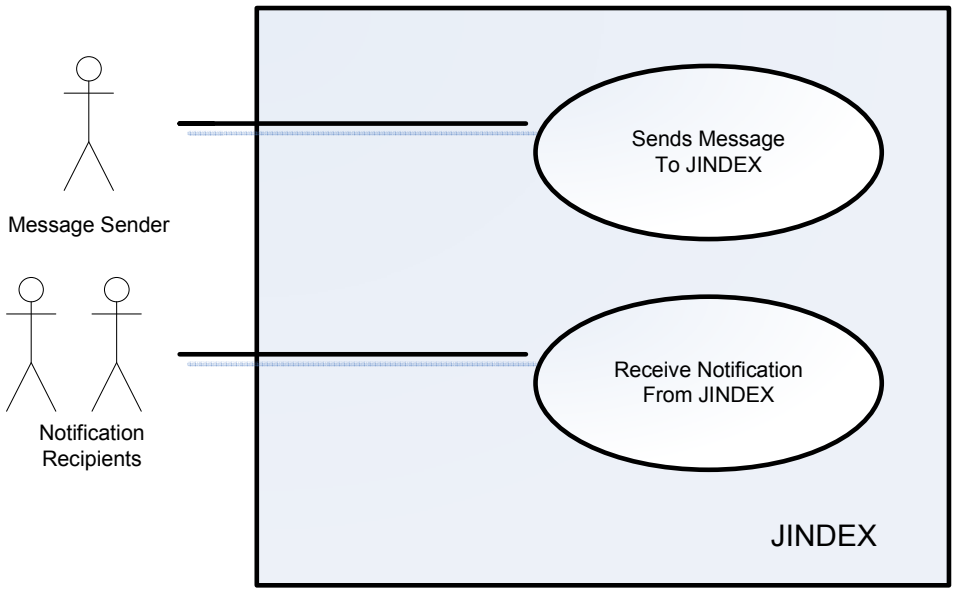
1.3.1 Sending Message in Normal or Test-Process Processing Mode from One Participant to Other Participants



1.3.2 Sending Notification from JINDEX to Participants



1.3.3 Sending Message in Test-Notify Processing Mode



2.0 Project Assumptions / Definitions

2.1 Technology Assumptions

This section identifies the conditions that impact the logical design. The following constraints and assumptions will be applied to this design.

Technology Selection

1. It is assumed that custom components may be written and deployed using Microsoft .NET Framework v2.0 and Microsoft Enterprise Library for .NET Framework 2.0.
2. It is assumed that the database platform will be Microsoft SQL Server 2000.
3. It is assumed that the message integration platform will be Microsoft BizTalk 2004.
4. It is assumed that the WS-* standards implementation will be performed using Microsoft WSE 3.0.
5. It is assumed that the notification procedures will be implemented using SQL Server Notification Services (SSNS).

Third-Party Components

1. It is expected that no third-party components will have to be purchased to implement this system.

2.2 Design Assumptions

1. The standard JINDEX/E-Trip message has two distinct parts; 1) The message *parameters* which are XML elements used to determine the sender, message type and routing itinerary, and; 2) The message *body* which contains the business content being exchange as an XML document. This design supports the requirement that stakeholders be able to modify the content (schema) of the message body without necessitating any modifications of the JINDEX messaging application
2. All six of the current message types (and any future message types) will conform to the standard JINDEX/E-Trip message construction ((1) set of parameters and (1) message body)
3. JINDEX will perform schema validation on the message *parameters*, but will not validate or perform any schema validation or transformation on the message body.

4. Stakeholder requirements dictate that digital attachments such as digital photographs or drawings will be contained within message body and will not separated from or processed differently in any way by the JINDEX platform.
5. Messages will be persisted in JINDEX until they are received and accepted (or rejected) by all Recipients.
6. After all Recipients respond to JINDEX, all original messages are removed from JINDEX for security reasons. JINDEX log files will keep only auditing, routing, and processing information.

2.3 Definitions

1. **Participant –** A state or local ***agency*** that is authorized to send or receive messages through the JINDEX platform. Throughout this document specific participants are referenced by the following abbreviations:
    - **DIS –** Washington State Department of Information Services
    - **AOC –** Washington State Administrative Office of the Courts
    - **WSP –** Washington State Patrol
    - **DOL –** Washington State Department of Licensing
    - **WSDOT –** Washington State Department of Transportation
    - **LEA –** Generic reference to a Law Enforcement Agency. WSP is the lead Law Enforcement agency on this project, but the term applies to any LEA who may become a participant in the future.
    - **LEO –** Generic reference to an individual Law Enforcement Officer.
2. **Message Sender –** The message sender is the agency that is sending the message. Within the JINDEX system, all messages are routed through the JINDEX platform in order to de-couple sending and receiving services. As such, the sender of a message is only the originator in half the cases. Example:
    - Law Enforcement sends a ticket to JINDEX. The sender is the law enforcement agency.
    - When JINDEX sends the ticket to AOC, JINDEX is the sender, though it is not the message originator.
3. **Message Originator –** This is the agency that originally created the message. It remains the same whether the LEA or JINDEX is the sender of the message
4. **Message Receiver** – A message receiver is any participant that receives E-TRIP messages as a part of a JINDEX Exchange. A Message Receiver can be an agency that is a consumer of a message, or can be the JINDEX platform which routes but does not retain messages

5. **Message Recipient –** This is the agency that is to receive and consume the message. There are no message types currently in scope for this project where JINDEX would be the end recipient.
6. **Message –** Also referred to as a ***JINDEX Document***, is a group of XML nodes (*Parameter*) and an XML document (*Message*) contained within a SOAP wrapper that is sent from one participant to another through the JINDEX platform. Each message must contain XML *Parameter* nodes, and an XML *Message* document.
7. **Acknowledgement –** Also referred to as a JINDEX Acknowledgement. This is a brief message returned to the participant who sent a message notifying the sender that their message was received. Acknowledgments are sent synchronously meaning that the sender will wait for an acknowledgement before sending another message. Acknowledgements can be either positive (ACK) or negative (NAK)

   (**Important note:** An acknowledgement only reflects the successful or unsuccessful communication of a message between JINDEX (as the sender or receiver) and another participant (as the sender or receiver). It does not indicate that a message has completed its routing itinerary and arrived at its final destination nor that it was processed successfully. An acknowledgement should not be confused with a *Functional Response Message* described below.
8. **JINDEX Exchange -** A term used to describe the interaction between the JINDEX messaging system and a *Participant* system. For example, an exchange is said to have occurred when a *Message* (JINDEX Document) is sent by an LEA to JINDEX, the message is successfully received by JINDEX, and JINDEX returns an acknowledgement (JINDEXAcknowledgement) for that message. For a message to be transmitted from one agency to another, two (2) JINDEX Exchanges must occur:
   - **Sending agency sends to JINDEX**
   - **JINDEX sends to receiving agency**

   For performance reasons, this concept is important to remember as a single Ticket message will result in four (4) JINDEX Exchanges, and a single Collision Message
9. **Original Message –** Also referred to as a Primary Message, is the first message that starts a document exchange. Within the scope of this project there are three Primary messages:
   - **TICKET_MESSAGE –** A message that originates from a Law Enforcement Agency and contains ticket data.
   - **COLLISION_MESSAGE -** A message that originates from a Law Enforcement Agency and contains Collision report data.
   - **DISPOSITION_MESSAGE –** A message that originates from AOC and contains information about a ticket that has been adjudicated by the courts.
10. **Functional Response –** A functional response (also referred to simply as a "response") is a separate and discrete message that is sent back to the

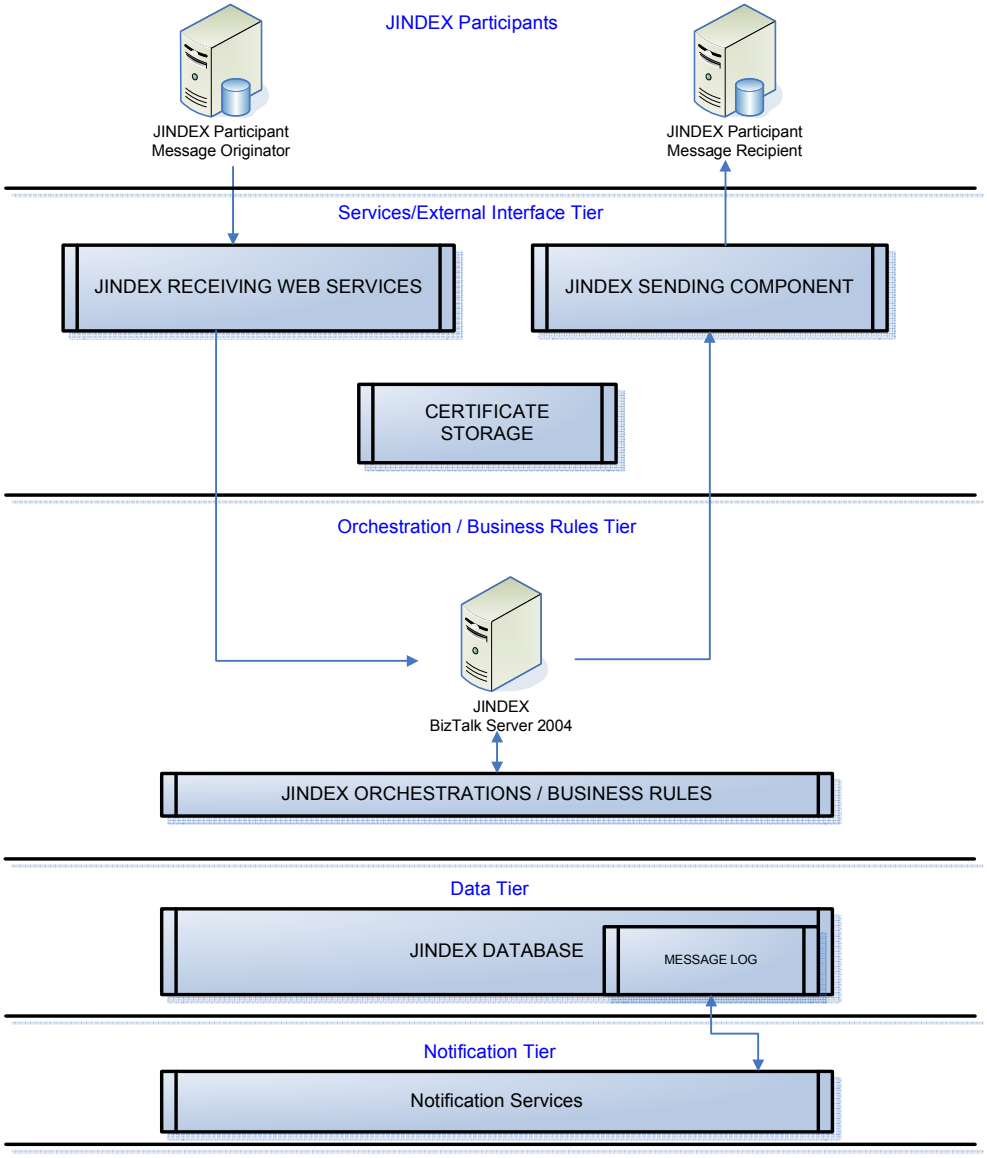
Message Originator by the *Message Recipient* confirming that their original message was received and processed.

An affirmative functional response will contain a receipt number (of some origin) like a Collision Report number or Court Case number. A rejecting functional response will NOT contain a receipt number and may contain textual information describing why the primary message is being rejected. All functional response are sent asynchronously, meaning that the Message Originator is not waiting for the functional response before it sends another message. The functional response can be returned near instantaneously, or can be delayed several hours without impacting the operations of the system. Within the scope of this project there are three Functional Response messages:

- **TICKET_RESPONSE –** A message that originates from AOC and is returned to the LEA that sent the original ticket message.
- **COLLISION_RESPONSE -** A message that originates from WSDOT and is returned to the LEA that sent the original Collision Report message.
- **DISPOSITION_RESPONSE –** A message that originates from DOL and is returned to AOC.

11. **Message Subscribers -** The JINDEX platform operates on a Publish / Subscribe model, meaning that any message type can have any number of subscribers. For example, when an LEA sends ("publishes") a collision report message to JINDEX, both WSDOT and DOL are configured as "Subscribers" to that message type and each receives a copy of the message.

3.0 Tiers

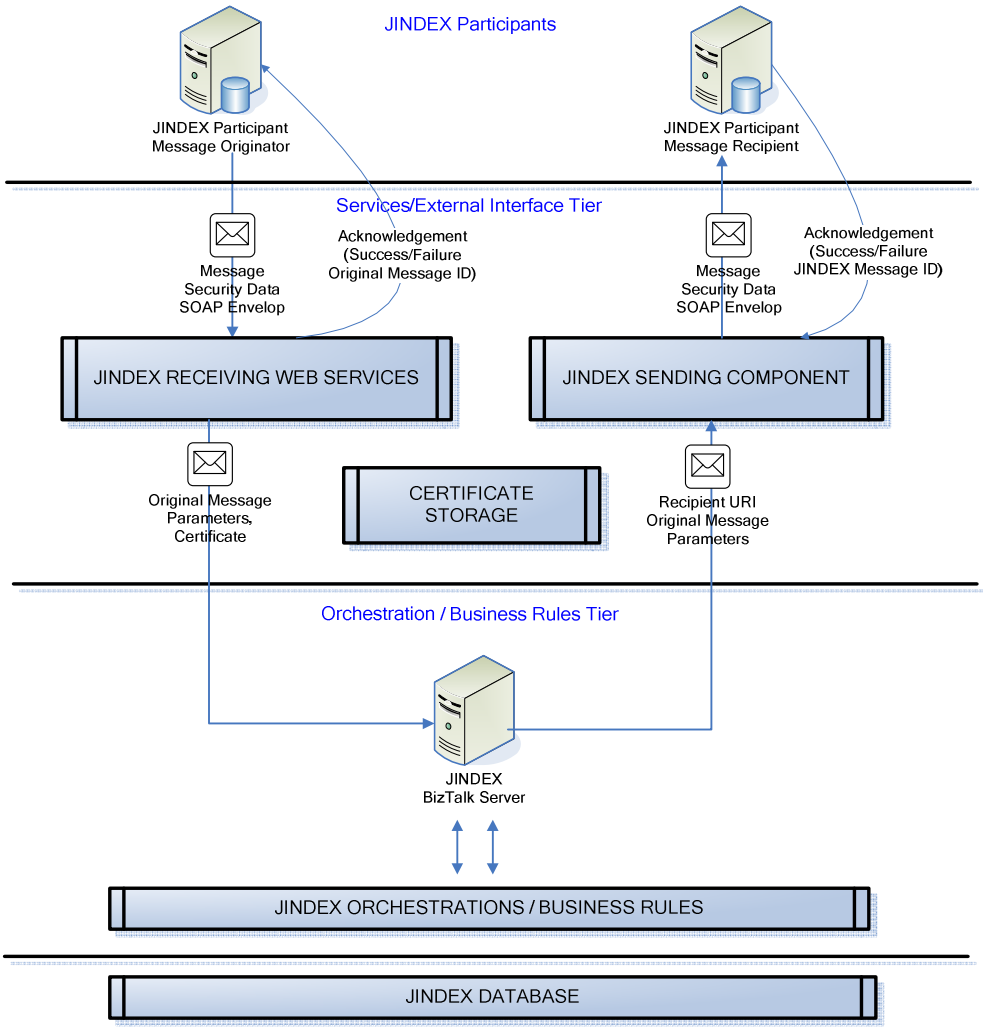

JINDEX Participants
JINDEX Participant
Message Originator
JINDEX Participant
Message Recipient
Services/External Interface Tier
JINDEX RECEIVING WEB SERVICES
JINDEX SENDING COMPONENT
CERTIFICATE
STORAGE
Orchestration / Business Rules Tier
JINDEX
BizTalk Server 2004
JINDEX ORCHESTRATIONS / BUSINESS RULES
Data Tier
JINDEX DATABASE
MESSAGE LOG
Notification Tier
Notification Services

4.0 Message Flow

4.1 Message Processing in Normal or Test-Process Modes (Use Case 1)

This case describes the message flow between one Participant (Message Originator) and other Participants (Message Recipients). Messages are sent from the first Participant to JINDEX, and then distributed from JINDEX to other Participants. To pass the messages all parties use push-technology rather than pull-technology. To pass a message to JINDEX, the Sender invokes JINDEX Web methods. To pass a message to Recipients, JINDEX invokes Recipient Web methods.

4.1.1 Message Flow Diagram



4.1.2 Message Flow Description

4.1.2.1 Receiving Message, Certificate Validation, Sender Authentication

1. Message Originator creates a message (example: traffic ticket, collision report, or a functional response) as an XML file, adds WS-* headers and submits the message to the JINDEX Receiving Web Service. The message is wrapped into a SOAP envelope and the WS-* section is added to the SOAP header. (If the Message Originator is based on the Microsoft platform, the suggested way to implement WS-* standards is WSE 3.0.)
2. The JINDEX Receiving Web Service receives the message, unwraps the WS-* section and performs the following actions based on the WS-* standards (not necessarily in the presented sequence):
    a. Validates Sender X.509 Certificate Structure;
    b. Decrypts the one time symmetric key with the JINDEX Private key;
    c. Decrypts the message with the one time symmetric key;
    d. Uses the Sender public key to validate the Sender signature;
    e. Checks that the Sender certificate is valid and matches an existing entry in the Trusted People folder in the Certificate Storage;
    f. Checks that the Certificate Authority is valid and matches existing entry in the Trusted Root Certification Authorities folder in the Certificate Storage;
    g. Checks that the Sender certificate is not revoked by the Certificate Authority.
3. The JINDEX Receiving Web Service validates the XML schema of the parameters.
4. The JINDEX Receiving Web Service extracts the parameters from the XML.
5. The Receiving Web Service parses the Sender X.509 certificate and extracts the Sender Certificate Number and the Issuer Name from the certificate.
6. The Receiving Web Service passes the message to BizTalk along with the Sender Certificate Number, the Issuer Name, and additional parameters.
7. In case of success, the Receiving Web Service returns an positive acknowledgement (ACK) to the Sender. The ACK contains a Return Code (P) and the Original Message ID.
8. If any error happened during the previous steps, the Receiving Web Service responds with a Negative Acknowledgement (NAK) to the Sender. The NAK contains a Return Code (F)ailure or a specific error code "E##") and the Original Message ID (assigned by the Sender). If the Original Message ID can

not be retrieved from the parameters, 0 (zero) is assigned to the Original Message ID.

4.1.2.2 Data Validation, Sender Identification, Sender Authorization, Original Message Schema Validation

1. BizTalk receives the message from the Receiving Web Service. The message Status field is set to P (Processing).
2. BizTalk validates the Message Type and Processing Mode fields against lookup tables.
    a. If the Message Type field is invalid, the message Status field is changed to E20 (Error) and the message information is stored in the JINDEX Message Log. *(NOTE: In release 1.10 and higher of the E-Trip Messaging system, Message Type is checked at the Web Service layer as an enumerated value in the WSDL. As such, E20 should never appear as a message status under normal conditions. Because of the critical nature of this element, the BizTalk logic still checks the validity of the value to ensure that the WSDL, and the routing logic stay in sync.)*
    b. If the Processing Mode field is invalid, the message Status field is changed to E22 (Error) and the message information is stored in the JINDEX Message Log. *(NOTE: In release 1.10 and higher of the E-Trip Messaging system, Processing Mode is checked at the Web Service layer as an enumerated value in the WSDL. As such, E22 should never appear as a message status under normal conditions. Because of the critical nature of this element, the BizTalk logic still checks the validity of the value to ensure that the WSDL, and the routing logic stay in sync.)*
3. BizTalk uses the Sender Certificate Number and the Issuer Name to retrieve the Sender ID from the participants table. BizTalk also populates the Sender ID element in the message.
4. If the Sender ID is not found in the participants table of the JINDEX database, the message Status field is changed to E24 (Error), and the message information is stored in the JINDEX Message Log.
5. BizTalk performs Sender authorization based on the Message Type and Processing Mode parameters to determine if the sender of the message is authorized to originate messages of the type submitted. (Law Enforcement may send tickets to AOC, but WSDOT may not… See the authorization tables in section 9). If the authorization fails, the message Status field is changed to E26 (Error) and the message information is stored in the JINDEX Message Log.
6. If any other exception occurred during the previous steps (1-5), the message Status field is changed to E50 (Error), and the message information is stored in the JINDEX Message Log.

4.1.2.3 Routing and Addressing

1. If the Processing Mode parameter of the message is Test-Notify, the message information is logged in the Message Log and BizTalk stops processing the message.
2. BizTalk determines the Recipients of the message. Other than Test-Notify, there are two possible routing mechanisms.
    a. With the exception of the Ticket Response and Collision Response messages, the Recipient(s) are identified by the Message Type and the Processing Mode parameters as described in the Routing section.
    b. In case of the Traffic Ticket Response and Collision Response, the JINDEXRecipientID element has to be present as one of the elements of the additional parameters.
        i. If the JINDEXRecipientID element is not present, the message Status field is changed to E30 (Error) and the message information is stored in the JINDEX Message Log.
        ii. If the JINDEXRecipientID element is present, it has to match one of LEA IDs in JINDEX participant database. In case it does not match a valid LEA ID, the message Status field is changed to E32 (Error) and the message information is stored in the JINDEX Message Log.
3. BizTalk retrieves the Current Recipient Status from the database.

    Note. Current Recipient Status field is set up by the system (JINDEX) administrator. The system administrator can modify the status any time on the request of the Participant and following correct procedures. The Not Responding status can be also set up by JINDEX if a call to the Recipient Web Services fails. A separate scheduled process (a SQL Server stored procedure) will change Not Responding statuses back to Normal status periodically.
4. For each Recipient, BizTalk determines the destination URI based on the Message Type, Processing Mode, Recipient ID, and Current Recipient Status fields as described in the Addressing section.
    a. The abovementioned combination of parameters may not result in any valid destination. In that case, the message Status field is changed to E34 (Error) and the message information is stored in the JINDEX Message Log.
    b. The Current Recipient Status may indicate that the message should be put on hold (Current Recipient Status = Hold, Current Recipient Status = Test and Processing Mode = Hold, Current Recipient Status = DRBC and Processing Mode = Test). In that case BizTalk will wait for 5 minutes then continue from step # 3 above.
    c. Current Recipient Status may indicate that the recipient is not responding. In that case BizTalk continues after a delay. The duration of the delays may be different for this and the previous cases.

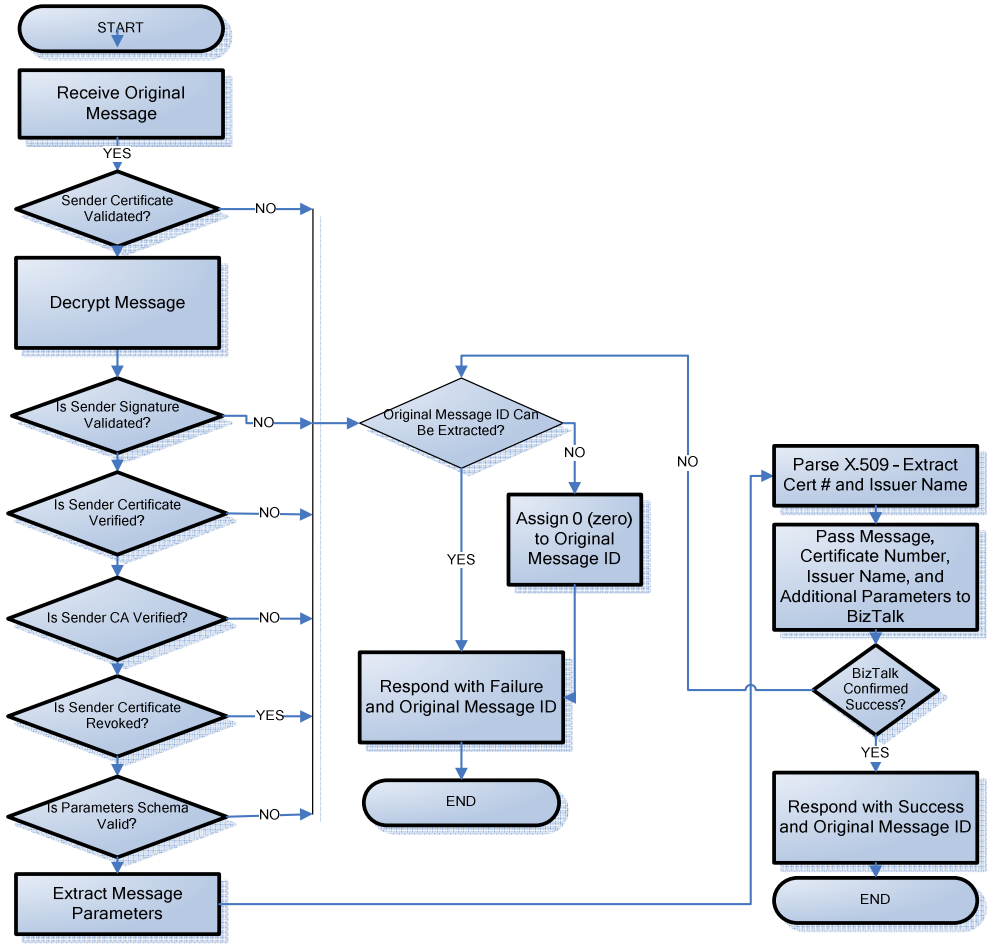
5. If the message is not put on hold and the destination is valid, BizTalk calls the Sending component and passes the message, parameters, and the Recipient URI to the Sending component.
6. If the Sending component responds with a communication error, the Current Recipient Status is changed to Not Responding. BizTalk continues from step # 5 after a delay.
7. If the Sending component responds with success, the message Status field is changed to "P" which indicates that the message exchange was Completed; the message information is logged in the Message Log; and the BizTalk orchestration concludes the processing of that message.
8. If the Sending component indicates the message was not accepted by the Recipient, the message Status field is changed to F (Failure), the message information is logged in the Message Log, and BizTalk stops processing the message.
9. For each Recipient steps #3-8 are repeated.

4.1.2.4 Sending Messages

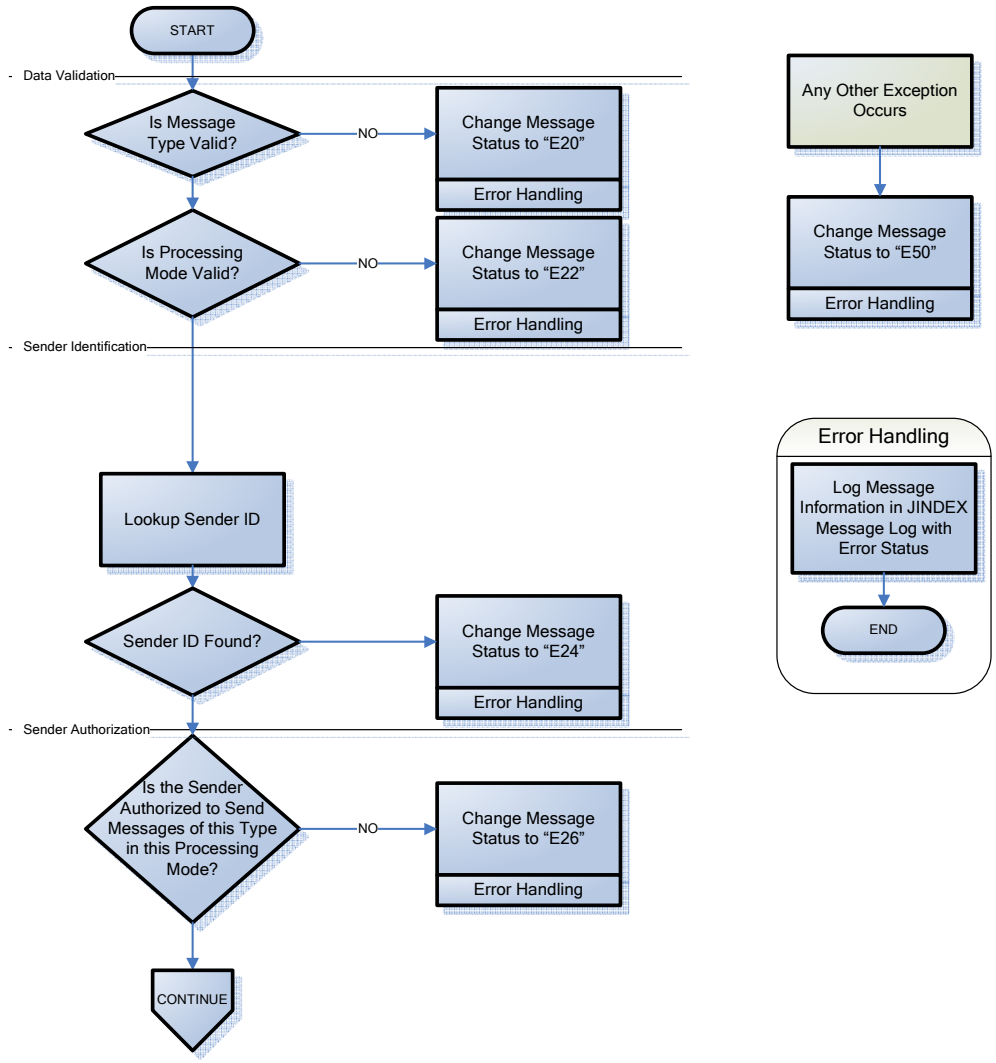
1. The Sending Component receives the message from BizTalk along with the parameters, and the Recipient URI.
2. The Sending Component retrieves the Recipient and JINDEX X.509 certificates from the Certificate Storage.
3. The Sending Component adds WS-* information, including JINDEX certificate, to the message and submits the SOAP wrapped message to the recipient URI.
4. The Recipient may respond with success or failure or a communication error may occur. In all cases, the Sending Component responds to BizTalk indicating whether the message was successfully delivered or not.
5. In case of communication error the Sending Component may attempt to resend the message one or more times based on the configuration parameters.
6. If any exception occurs during steps 1-5, the Sending Component responds to BizTalk indicating the error.

4.1.3 Logical Flow Diagram

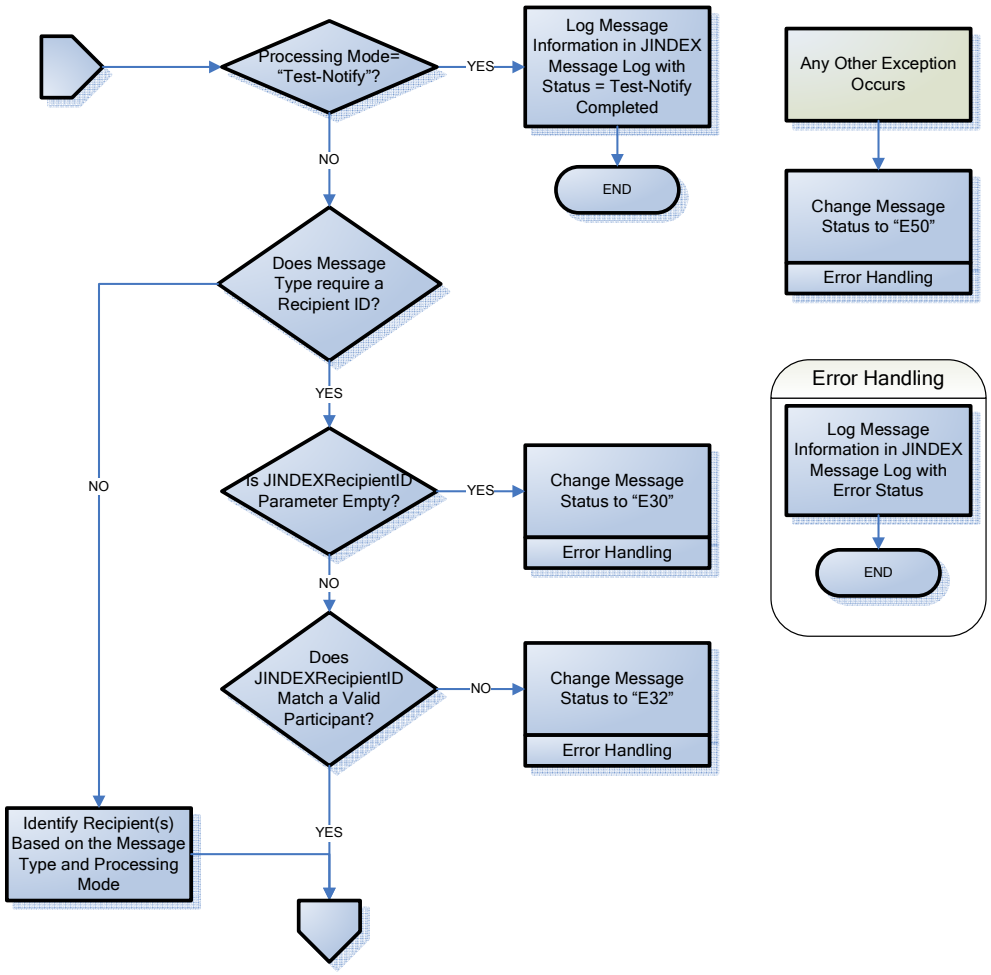
4.1.3.1 Message Receiving, Certificate Validation, Sender Authentication



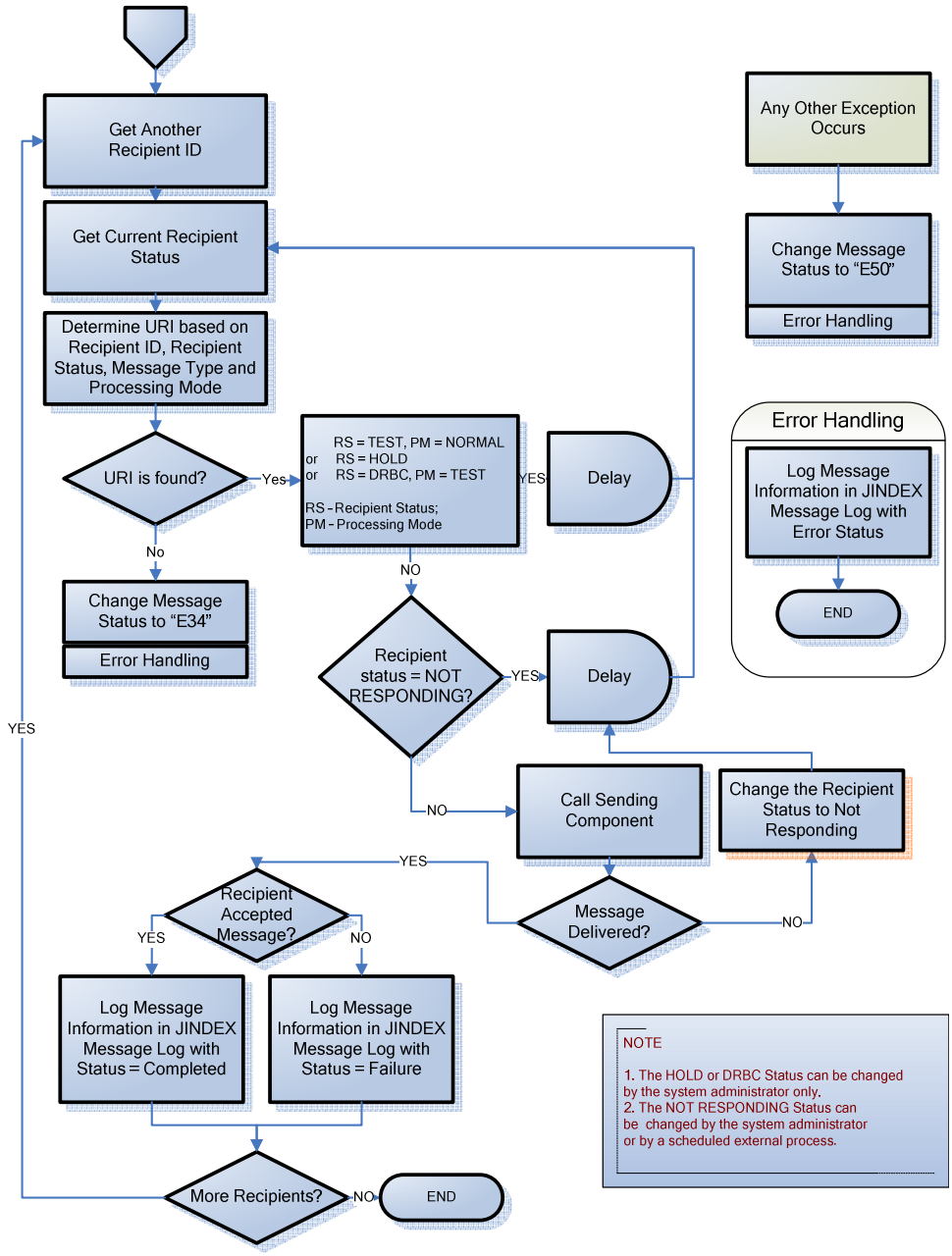
4.1.3.2 Data Validation, Sender Authorization



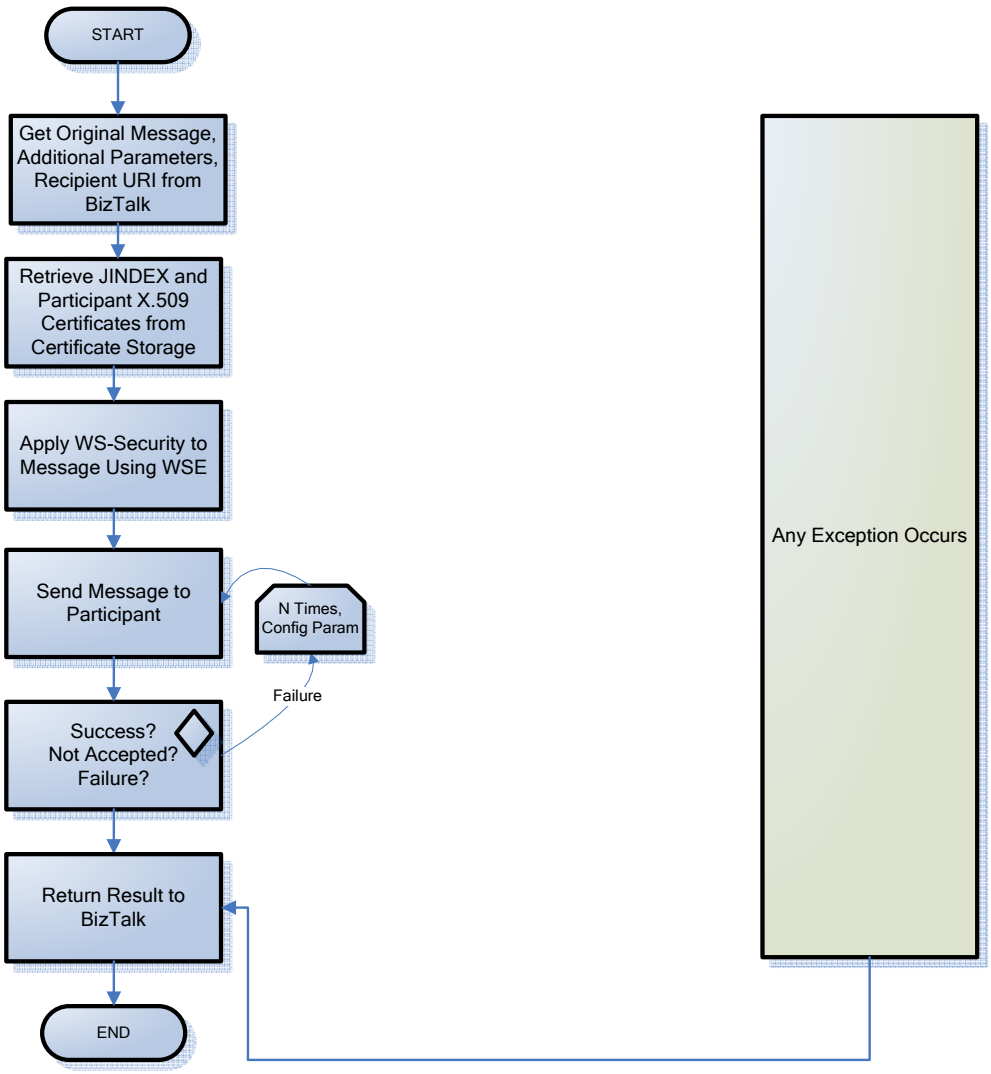
4.1.3.3 Routing



4.1.3.4 Addressing

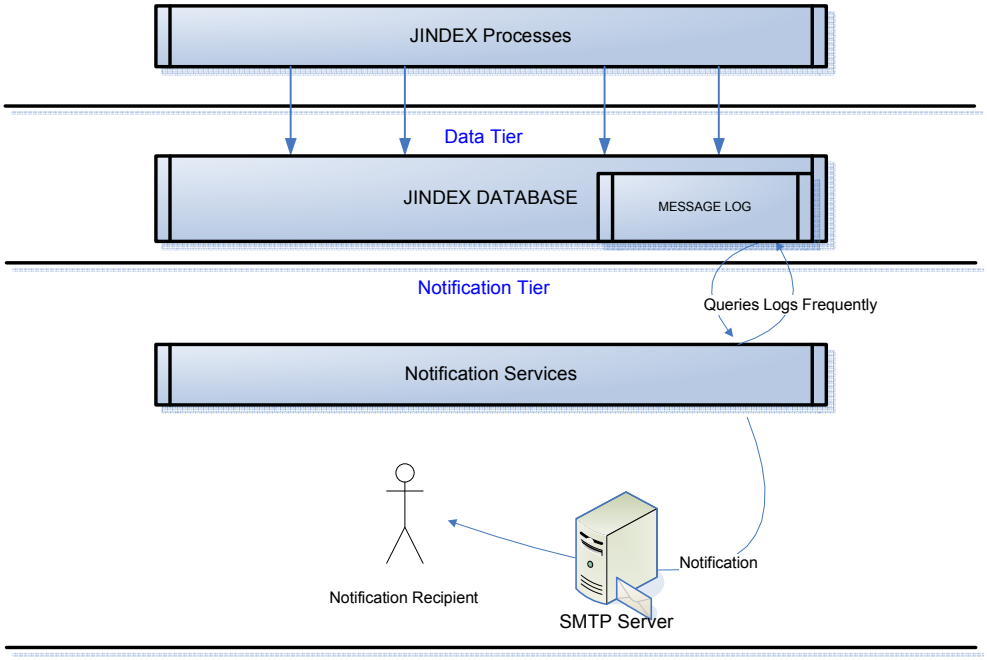


4.1.3.5 Sending Messages



4.2 Notifications (Use Case 2)

4.2.1 Message Flow Diagram



4.2.2 Message Flow Description

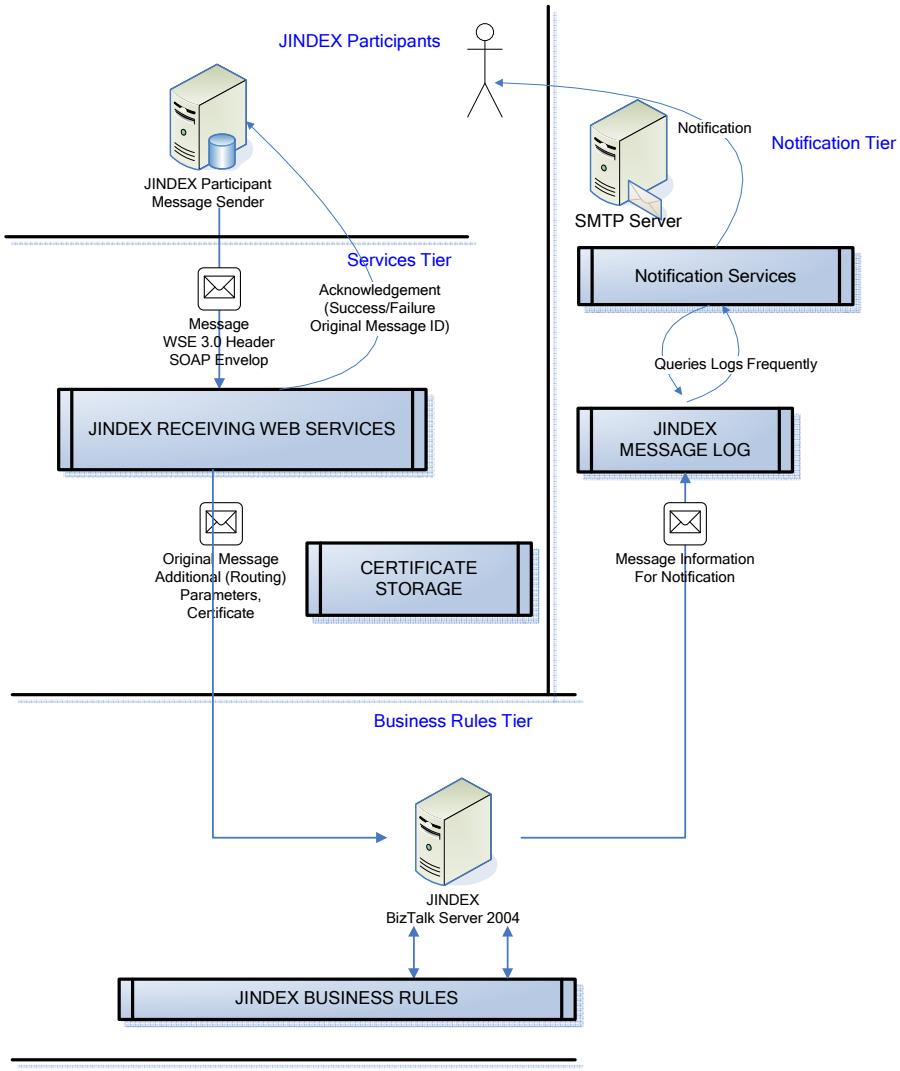
1. Different JINDEX processes may encounter errors or need to issue notifications to JINDEX Participants (i.e. messages sent in the Test-Notify mode).
2. These processes call a SQL stored procedure to store this information in the JINDEX Message Log.
3. MS SQL Server Notification Services query the Message Log periodically to find new records.
4. Notification Services determines recipient(s) of the notification message based on their subscription to notifications.
5. In case of a new record, Notification Services will create a new notification message (e-mail).
6. Notification Services submit the notification to an SMTP server for delivery.
7. Notification Services update the notification record indication that the notification was sent.

4.3 Test-Notify Messages (Use Case 3)

Message will be processed through the entire JINDEX – E-TRIP system, but will not be forwarded to the subscribing Participant(s). After completing the Business Rules layer of the process, a record will be placed in the Message Log indicating successful processing of the record. The Notification Services will send an e-mail to an appropriate contact for the originating Participant indicating the successful arrival of the message.

This use case is a combination of use cases 1 and 2.

4.3.1 Test-Notify Message Flow Diagram



4.3.2 Test-Notify Message Flow Description

Test-Notify messages are received according to the description in section 5.1.2 (items 1-17). However, instead of sending the message the following steps are performed:

1. The message Status is changed to T ("Test-Notify completed").
2. The message information is placed in the Message Log.

3. BizTalk stops processing the message.
4. The notification is sent to the destination as described in section 5.2.2.

5.0 Security Model Overview

The security model for JINDEX is comprised of three complementary security elements:

- **Authentication –** The methods and processes used to reliably determine the identity of a user of the JINDEX system. The authentication model answers the questions "Who is this user?" and "Are they a valid JINDEX participant?"
- **Authorization –** The methods and processes used to determine which services of JINDEX are available to valid participants. For example, Law Enforcement agencies are able to submit tickets and collision reports to JINDEX, but may not submit a Court Disposition report to the Department of Licensing
- **Data Privacy –** The methods and processes used to ensure that sensitive data remains protected while it is being transmitted from one participant to another (Transport Security) and while it is being stored (persisted) at a participating partner. (Secure Persistence)

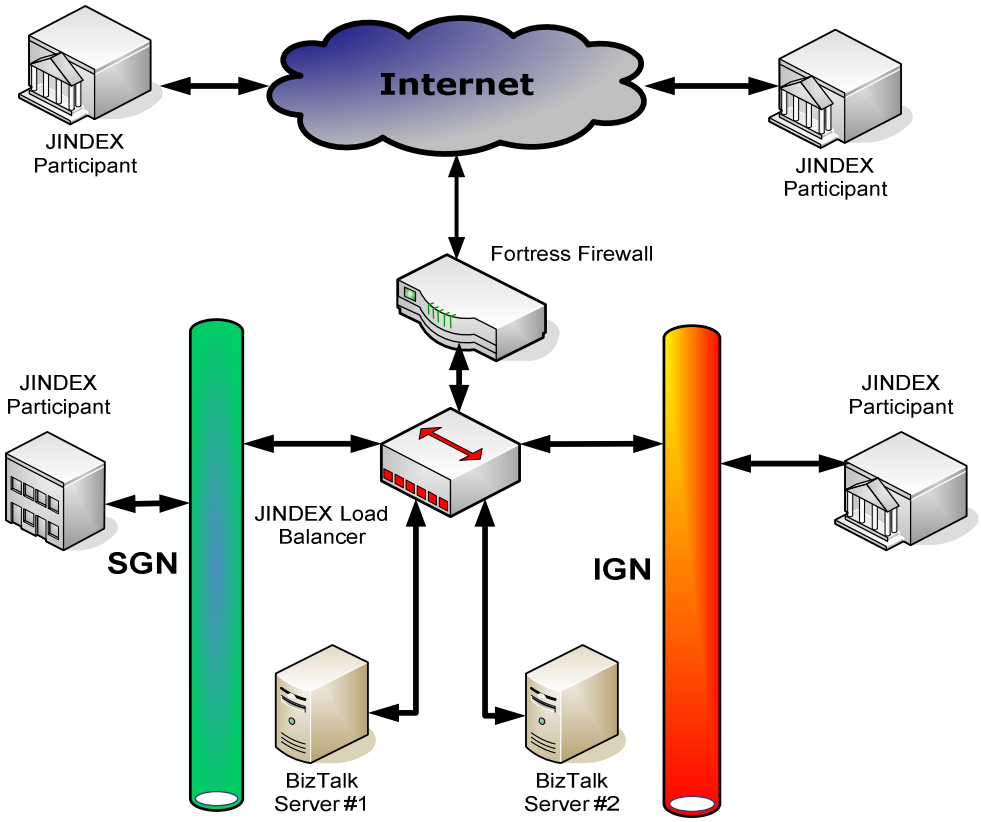
Recommendations and minimum standards for each of these elements are detailed in the *Specification section* of this document.

Communication Networks

The Washington Department of Information Services (DIS) manages the voice and data communications networks for a large percentage of State and Local Governmental entities. Within the justice ecosystem, three networks are used to communicate with other JINDEX participants and are depicted below:

- **State Governmental Network (SGN) –** A high speed, secure private network for exclusive use by State Agencies.
- **Intergovernmental Network (IGN) –** A broadly deployed private network used by State Agencies, Local Governmental entities and Law Enforcement.
- **The Public Internet –** Used by the public at large, and a growing number of local entities and State Agencies

The security specifications being proposed will provide a ubiquitous level of security and reliability regardless of which network is used.



The State of Washington requires that any State agency accessing the public internet must pass through a security filter / firewall known as Fortress. Fortress is a hardware / software based system that separates the States "internal networks" (SGN & IGN) from the "external" public internet. Because some participants need to access JINDEX via the internet, all of their traffic must pass through the Fortress system. Those participants that are connected to either the IGN or SGN do not have this requirement.

One of the primary objectives of this design is to provide a security model that can be applied to all message traffic such that a single set of policies can be developed and enforced regardless of which network carries the message traffic.

5.1 Security Specifications – Overview

Several possible security models were considered as the following specifications were being developed. In each case, we considered the following factors while evaluating each model:

- Highly secure
- Scalability
- Architectural flexibility
- Standards based
- Network Independence
- Topology neutral
- Forward looking industry acceptance
- Ease of implementation
- Low maintenance
- Universally applicable to current and future JINDEX participants

The two basic Security models considered were:

- **Transport Layer Security** model using Secure Sockets Layer (SSL) which creates a secure point to point pipeline.
- **Message Layer Security** model utilizing Web Services extension specifications like Web Services – Security (WS-S) and Web Services Secure Conversation (WS-SC). "Message Level Security"

After considering both models, it was determined that the JINDEX would be based on Message Layer Security. This means that rather than creating secure channels for the transmission of clear text data, each message will be encrypted using a digital certificate and sent within a standard SOAP envelope.

The following sections describe at a high level the functioning of JINDEX within the proposed Security Model, within the context of the three security elements discussed previously 1) Authentication; 2) Authorization; 3) Data Privacy.

5.2 Authentication

Authentication seeks to determine; 1) who is sending a message to JINDEX, and 2) that the party on the receiving end of a message is who JINDEX thinks they are.

The authentication portion of the JINDEX Security Model (JSM) will be accomplished through the use of X.509 v3 Digital Certificates issued through a third party Certificate Authority (CA). As of October, 2006, the State of Washington has a special program with Digital Signature Trust (DST) (http://www.digsigtrust.com/state/wa/) as their preferred Certificate Authority, however identical certificates are available from other certificate authorities.

JINDEX participants may acquire a TrustID Server Certificate from DST for $175/year (2006). This certificate will be used to secure and authenticate messages originating from or being sent to a web or network server or JINDEX.

TrustID certificates are administered under the TrustID Certificate Policy sponsored by the American Bankers Association. TrustID server certificates are standard x.509

v3 certificates and work with WS-* message security standards as well as SSL-capable browsers and servers.

DST issues TrustID Server Certificates to *agencies* that are identified and authenticated during online registration, as well as by third-party proofing and out-of-band notification. As a part of the certification process, JINDEX participants will receive a Public key as well as a Private key. The public key will be shared with JINDEX in a secure, out of band process, but JINDEX participants are expected to use reasonable efforts to protect the data and physical security of their Private Key. A TrustID Server Certificate verifies the identity and ownership of a network server and will be used to encrypt individual messages traveling to and from JINDEX.

5.2.1 Authentication Process

Every JINDEX partner complying with this specification will be required to obtain an X.509 v3 digital certificate. JINDEX partners will share their Public Keys with JINDEX, and JINDEX will share its Public Key with each of the participants. JINDEX and each of the participants will ***NEVER*** share their private keys.

Whenever a message is sent, it is digitally signed using the sender's private key and encrypted using the destination's public key. A copy of the sender's public key is sent along with the message.

When the message is received, the receiving server will:

- Validate the integrity of the message received using the senders public key (data has not been corrupted)
- Compares the public key to the copy of the public key that was previously shared with JINDEX.
- Extract unique identifying information from the certificate and perform a table look-up returning a valid participant ID.

Should any of these tests fail, JINDEX will terminate the session, log the error, discard the message and optionally, initiate a notification sequence to JINDEX operations and the DIS Network Security Department.

If the message passes all of these tests, the message is determined to "Authenticated", which means it came from a legitimate JINDEX participant.

> ***NOTE:*** *WS Secure Conversation (WS-SC) allows for JINDEX and a participant to set up a single "conversation" or session whereby a single encryption/decryption key is used to secure a group of messages sent in a "burst". This is a performance boosting feature that will be employed as a part of securing the message to and from JINDEX. The specifics of WS-SC will not discussed in detail within this document.*

5.3 Authorization

After the "Authentication" process has reliably determined ***who*** the sending participant is, the Authorization process will determine ***what*** message types the participant is eligible to send. Authorization will be accomplished via a look-up to a database table as the message passes through the JINDEX system. This will be a standardized process that will initially be used by the JINDEX-Traffic Records project, but can be utilized by any subsequent project that follows standard BizTalk messaging patterns.

5.3.1 Authorization Process

As the message enters the JINDEX platform, the Authentication processes will, using the digital certificate included with the message, positively identify who the sending party is and that they are a valid user of the JINDEX platform.

Unique information contained within the certificate will be used to do a look-up from a participants table that will return the Participant ID. This value will be inserted into the OriginatorID field in the message header, overwriting any value present.

Imbedded in the message header will be the text element "MessageType", which will have (SECTOR) values like "TICKET_MESSAGE", "COLLISION_MESSAGE" or "TICKET_RESPONSE". Using the OriginatorID and the MessageType, the authorization process will do a look-up into the Authorization database table and retrieve the authorization rights for the sender / message being presented. Possible levels of authorization will be:

- **Normal** – The sender's messages have been previously certified and are permitted to be published to the ***production*** environments of subscribing participants.
- **Test-Process –** Messages will be processed through the entire JINDEX – E-TRIP system as if production data. When the message passes through the Routing Function, the destination address will be set to the destination ***Test*** Environment URI.
- **Test-Notify –** Message will be processed through the entire JINDEX – E-TRIP system, but will not be forwarded to the subscribing participant(s). After completing the Business Rules layer of the process, a record will be placed in the notification table indicating successful processing of the record. The notification system will send an e-mail to the Development Contact for the originating participant indicating the successful arrival of the message.
- **BLOCKED -** The authenticated Sender is not allowed to send this message type. The message will be dropped and an e-mail with the senders message ID will be sent back to the sender and a copy sent JINDEX Security officer.

The authorization table will contain "true/false" columns for each of these processing modes. Within the header of the message will be a "ProcessingMode" element whose value will be one of the following:

- **Normal**
- **Test-Process**
- **Test-Notify**

The message is considered authorized, when the ProcessingMode Element is equal to one of the above strings, AND, the corresponding authorization element is set to "True".

If a message is received from an authorized sender, that does not have a corresponding row in the authorization table, the message is treated as **Blocked.**

The schema for the authorization table is:

Key?	Column Name	Description	Data Type	Acceptable Values
YES	ParticipantID	Unique integer identifier for a participant (Message Originator)	Integer	(1-99999999)
YES	MessageTypeID	Type of message being sent	Text	(ex. TICKET_MESSAGE; COLLISION_MESSAGE; COLLISION_RESPONSE)
YES	ProcessingModeID	Processing Mode	Integer	1=Normal 2=Test-Process 3=Test-Notify
	AvailableYN	Is combination of ParticipantID, MessageTypeID, and ProcessingModeID authorized?	Logical	True / False
Note: Each participant will have a record in the authorization table for each message type / processing mode combination. If a matching record does not exist, the system will block authorization by default.				

Individual exchanges may choose to implement some, or all of the above authorization functionality. Minimum functionality of the authorization process will be to treat the message as Normal, or Blocked.

5.4 Data Privacy

Data Privacy is the third and final element of the proposed Security Model and covers the requirements of securing the message data in transit to and from JINDEX, and securing the data while it is stored, either within JINDEX, or within the sending or receiving participant datacenter. (Also known as "Persisted Data")

5.4.1 Data in Transit

As mentioned previously, all messages transmitted to and from the JINDEX platform will be encrypted using a 128 bit key associated with the digital certificates of the sending and receiving parties. Each message will be encrypted by its sender, and decrypted only by the receiver of the message regardless of how many intermediary devices the message passes through. (i.e. Fortress)

5.4.2 Persisted Data

Messages and data that are stored within a file server or on a laptop computer in a Patrol Officers Squad car are examples of persisted data. This model proposes two basic rules for persisted data:

1. If the device that contains the messages or data is easily portable, or typically resides in an unsecured vehicle or facility, then the messages and data on that device are considered vulnerable and should be encrypted on the data storage mechanism of the device. A laptop computer in a police squad car is an example of a vulnerable data store.
2. If the storage device that contains the messages or data, resides within a secure facility or data center that maintains active and rigorous control over access to the device and the data stores, then the data is not considered vulnerable, and no additional data security should be required. Determination of a facilities vulnerability status will be documented in the Service Level Agreement between the participating participant and DIS.

5.4.3 Certification Protocol

As new message exchanges and new participants are brought online using the JINDEX platform, a certification protocol should be followed. Though not strictly part of the technical security model, the certification protocol is an important procedural component of implementing the security model. The following points should be included as a part of the participant certification process:

- Participant enters into a Service Level Agreement with DIS (JIN) covering their usage of the JINDEX platform. The SLA will include individual Operation Level Agreements covering:
    - Datacenter Security and availability
    - Network Capacity, Availability, Security and Support
    - Server Capacity, Availability, Security and Support
    - Application Support & Change Control
    - Message Exchange Agreements (MEA) that cover the content, usage, exchange protocol and specific security requirements of the

information being exchanged. (One SLA can have many MEA's associated with it.

- Participant acquires an x.509v3 digital certificate from a Certificating Authority. The Participant will provide JINDEX with its Public Key.
- Participant successfully completes a series of scripted test routines ensuring that they are exchanging data in compliance with the specific MEA being certified.
- Responsible Governance board (E-TRIP etc) approves the participant for production usage.
- WIJIB approves the participant for usage.
- JINDEX operations team updates the Authorization table reflecting production status

6.0 Web Services Enhancements

6.1 WS-Security

WS-Security is a standard specification that describes enhancements to SOAP messaging to provide message integrity and confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

The JINDEX design is based on the following standard specifications:

Web Services Security: SOAP Message Security 1.0 (OASIS Standard 200401, March 2004).

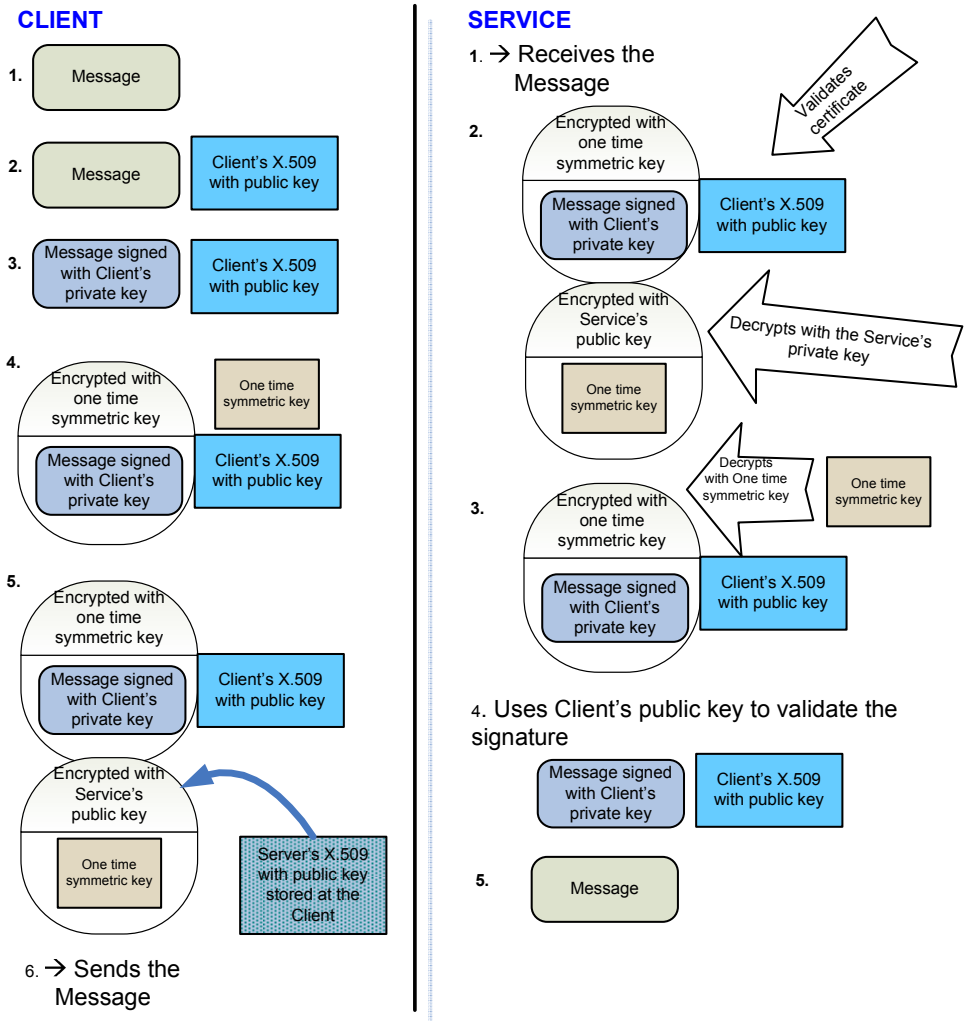
Web Services Security: X.509 Certificate Token Profile (OASIS Standard 200401, March 2004).

Web Services Security: Username Token Profile 1.0 (OASIS Standard 200401, March 2004).

6.1.1 Sending Secure Message

This section describes proposed sequence of steps to be implemented for each client and service to conform to WS-Security standards. The WS-Security standards will be implemented using WSE 3.0.

6.1.1.1 Diagram



6.1.1.2 Description

6.1.1.2.1 The Client Initializes and Sends a Message with X.509 Certificate Information

This part of the process has six steps:

1. The client retrieves the service's X.509 certificate.
2. The client retrieves its own certificate and private key.
3. The client attaches its X.509 certificate to a message.
4. The client signs the message using its private key.
5. The client encrypts the message using the service's public key.
6. The client sends the message to the service.

Step One. The Client Retrieves the Service's Certificate

The client needs to access the X.509 certificate of the service to encrypt the request message. The WSE 3.0 policy assertion on the client is configured to retrieve the service's certificate from the client's local certificate store without the need for any additional code.

Step Two. The Client Retrieves Its own X.509 certificate and Private Key

The client accesses its X.509 certificate and private key. It uses the private key to sign the message and the X.509 certificate to provide the service with the public key and other information about the client for verification with the service.

Step Three: The Client Attaches Its X.509 Certificate to a Message

WSE 3.0 policy is configured to sign the message, and WSE 3.0 automatically attaches the client's certificate to the request message.

Step Four: The Client Signs the Message Using Its Private Key

The client uses its private key to sign the message. One can choose to sign one or more portions of the message, such as the address header or the message body. At a minimum, one should sign the message body, security, and addressing headers. A signature is created using a signature algorithm that computes a checksum value from the data to be signed and then encrypts the checksum value with the client's private key. When the signature is validated, the data used to create the signature is also validated to provide data origin authentication.

Step Five: The Client Encrypts the Message Using the Service's Public Key

One can encrypt message parts using a symmetric key that is encrypted with the public key from the service's X.509 certificate. At a minimum, the signature used to sign the encrypted data is itself encrypted to help protect it against offline attacks.

When one uses WSE 3.0 policy to encrypt message data with X.509 certificates, the policy uses asymmetric encryption to encrypt a one-time symmetric key, which in

turn encrypts the data. When message data is encrypted using the service's certificate information, WSE 3.0 also adds the certificate identifier to the message. If the certificate contains a subject key identifier, this is included to identify the certificate in the message. Otherwise, the policy uses the issuer name and certificate serial number instead. The service owns the certificate, which contains all the necessary information for it to access the appropriate private key and decrypt the symmetric key, which is then in turn used to decrypt the message.

Encrypting the request in this way protects sensitive data if the client is deceived into calling an illegitimate service. As the intended message recipient, only the correct Web service can decrypt the message with its private key.

Step Six: The Client Sends the Message to the Service

After the message is signed and encrypted, the client sends it to the service.

6.1.1.2.2 Service Authenticates a Client Using the X.509 Certificate and Signature

This part of the process has six steps:

1. The service validates the client's certificate.
2. The service verifies the certificate trust chain.
3. The service checks the certificate revocation status.
4. The service decrypts the message.
5. The service verifies the signature.
6. The service initializes and sends a response to the client (optional).

Step One: The Service Validates the Client's Certificate

WSE 3.0 validates the client's certificate attached to the request message. The certificate's validity period is checked to ensure that the service does not process a request that was secured with an expired X.509 certificate.

WSE 3.0 also verifies the integrity of the certificate's contents to ensure that it has not been tampered with after the certificate authority (CA) issued it. The integrity of the certificate's contents is verified using the signature of the issuing CA, which is also included in the certificate. If the certificate's contents cannot be validated against the issuer's signature, then the certificate has been tampered with and it is rejected as invalid.

Step Two: The Service Verifies the Certificate Trust Chain

By default, WSE 3.0 verifies the trust chain of certificates, or requires that the client's certificate is installed in the **Trusted People** folder in the service's local certificate store. WSE 3.0 must be able to recognize an issuing CA as trusted to verify the certificate trust chain for the client's X.509 certificate. WSE 3.0 recognizes an issuing CA as trusted based on the X.509 certificate that endorses the client's certificate. WSE 3.0 recognizes the issuing CA's certificate as a trusted root for a certificate chain if the CA's X.509 certificate is installed in the machine certificate store in the **Trusted Root Certification Authorities** folder.

The high-level steps to install a certificate chain are as follows:

1. Export the certificate chain from the CA. This is dependant on the type of CA that issued the certificate.
2. Import the certificate chain into a local certificate store.

Step Three: The Service Checks the Certificate Revocation Status

WSE 3.0 policy checks the revocation status of the certificate by verifying whether the certificate is on a certificate revocation list (CRL) that the CA publishes. One can obtain the CRL out-of-band by downloading it from a CA, and then importing it into a local certificate store where WSE 3.0 can access it. You can also check the revocation status of the certificate online. However, this approach relies on an online revocation service that the service must access to verify the certificate's revocation status. There is also a performance cost associated with checking the revocation status online. For this reason, one may want to consider downloading the CRL instead and frequently updating the cached CRL. By default, WSE 3.0 verifies the revocation status of X.509 certificates online.

Step Four: The Service Decrypts the Message

By default, the **mutualCertificate10Security** assertion protects the message body by encrypting it. When WSE 3.0 receives an encrypted message, WSE 3.0 policy automatically decrypts it using the following steps:

1. WSE determines the value to identify the service's certificate—either the RFC3280 Subject Key Identifier, or the issuer name and serial number—that the client included in the message tells the service which certificate was used to encrypt the message. WSE 3.0 policy uses this value to determine which private key it must use to decrypt the message.
2. WSE decrypts the asymmetrically encrypted, one-time symmetric key that the client sent with the message, using the service's private key
3. WSE uses the symmetric key to decrypt the message data using a symmetric algorithm. By default, WSE 3.0 uses AES 256 for symmetric encryption.

Note Service side policy alone does not stop a client from sending an unencrypted message. However, policy will reject a message at the server if it is not encrypted.

Step Five: The Service Verifies the Signature

WSE 3.0 verifies the client's signature on the incoming request message using the public key sent with the message. If the message data is signed, this step also validates the client as the message originator to provide data origin authentication.

Step Six: The Service Initializes and Sends a Response to the Client (Optional)

If the service returns a secure response to the client, the same process described in these steps is used for the response message between the service and the client, except that the roles of the client and the service reverse. However, unlike the request message, the service does not attach its X.509 certificate to the response message, because the client already has a copy of it.

Instead, WSE 3.0 policy adds a reference to the service's certificate in the response message. The service initiates and sends the response, signs it with the service's private key and encrypts it with a symmetric key that is encrypted with the client's

X.509 certificate public key. The client processes the response in the same manner as the service processed the request: decrypt the symmetric key with the client's private key, and then decrypt the encrypted message parts with the symmetric key. Finally, the client verifies the service's signature with the service's X.509 certificate.

6.1.1.3 Implementation

Implementation of the processes in JINDEX will be done using MS WSE 3.0. Suggested scenarios are described in ***Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE 3.0)*** by Microsoft.

6.2 WS-Secure Conversation

WS-Secure Conversation is a standards specification that defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts. It provides secure communications across one or more messages.

JINDEX design is based on the following standard specification:

Web Services Secure Conversation Language (WS-SecureConversation).

6.2.1 Secure Conversation Design

This section describes proposed sequence of steps to be implemented for each client and service to conform to WS-Secure Conversation standards.

6.2.1.1 Description

There are several reasons for establishing a secure conversation, including:

- Preventing the client from having to present a user name and password each time it accesses a different service. This could involve the client having to cache the client's original credentials (which is not considered a safe security practice) or prompting users to provide their credentials each time.
- Improving performance when resource-intensive forms of credentials, such as X.509 digital signatures, are used. Creation and validation of X.509 digital signatures is a computationally intensive process, so performance can be improved if they are used less frequently.

WS-SecureConversation is a Web service specification that builds on WS-Security and WS-Trust. It describes how to establish a lightweight security context between two parties. The security context uses session keys; these session keys become the basis for encrypting and signing subsequent message exchanges, which results in more efficient secure communications between the two parties.

Note The security of any conversation depends on the key exchange mechanism. Typically, the key exchange mechanism is based on a key management infrastructure, such as one based on PKI or shared secrets.

6.2.1.2 Implementation

Implementation of the processes in JINDEX will be done using MS WSE 3.0.

7.0 Web Services Interface

7.1 General Description

Messages are passed between Participants and JINDEX. To pass the messages all parties use push-technology rather than pull-technology. To pass a message to JINDEX, the Participant invokes JINDEX Receiving Web Service. To pass a message to a Participant, JINDEX invokes the Participant Receiving Web Service. The Web Service Description and message format are identical among all participants, though some participants may enforce some business rules differently than others.

Each message sent will be acknowledged with success/failure synchronously, in receiving messages is reported to the Sender .

7.2 Receiving Web Service

7.2.1 JINDEXExchange Web Method

All JINDEX /E-Trip participants support the JINDEXExchange Web method. Participants call this method synchronously to submit documents to JINDEX. A completed exchange involves sending a message and receiving back an acknowledgement, The input message is named "JINDEXDocument" and the acknowledgement message is named "JINDEXAcknowldegement". There can be one and only one JINDEXDocument coupled with one and only one JINDEXAcknowldegement.

The JINDEXDocument contains two substructures named JINDEXParameter and JINDEXMessage. JINDEXParameter contains eight (8) elements used to validate the authorization of the sending participant and reliably route each message to its correct destination. These eight elements are defined explicitly in the web service description provided below (see WSDL listing). The JINDEXMessage structure contains the justice related XML data being shared by the participants. The actual schema of the JINDEXMessage is not defined in the web service description and is left to the sending and receiving participant to jointly develop and maintain. Because of this design feature, participants may enhance and expand the content of the JINDEXMessage without concern about it affecting the core functions of the JINDEX / E-trip messaging system.

7.2.1.1 Signature

Target namespace: "http://WSDIS.eTrip.ReceivingServices/"

```
public XMLDocument JINDEXExchange
    (XMLDocument JINDEXDocument)
```

7.2.1.2 Input Document Formats

Messages are transmitted between participants within the context of a complex XML document structure called JINDEXDocument.

JINDEXDocument

Below is a portion of the actual WSDL for the JINDEXExchange method describing the elements of the JINDEXDocument.

```
<s:schema elementFormDefault="qualified" targetNamespace="http://WDIS.E-Trip">
    <s:element name="JINDEXDocument" type="s1:JINDEXDocument" />
    <s:complexType name="JINDEXDocument">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="JINDEXParameter" type="s1:JINDEXParameters" />
        <s:element minOccurs="1" maxOccurs="1" name="JINDEXMessage">
          <s:complexType mixed="true">
            <s:sequence>
              <s:any />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>
    </s:complexType>
    <s:complexType name="JINDEXParameters">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="MessageType" type="s1:MessageTypes" />
        <s:element minOccurs="1" maxOccurs="1" name="ProcessingMode" type="s1:ProcessingModes" />
        <s:element minOccurs="1" maxOccurs="1" name="CreateDateTime" type="s:dateTime" />
        <s:element minOccurs="1" maxOccurs="1" name="OriginatorID" type="s:int" />
        <s:element minOccurs="1" maxOccurs="1" name="OriginatorMessageID" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="PersonalIdentifier" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="JINDEXRecipientID" type="s:int" />
        <s:element minOccurs="0" maxOccurs="1" name="JINDEXMessageID" type="s:int" />
      </s:sequence>
    </s:complexType>
    <s:simpleType name="MessageTypes">
      <s:restriction base="s:string">
        <s:enumeration value="NULL_MESSAGE" />
        <s:enumeration value="COLLISION_MESSAGE" />
        <s:enumeration value="COLLISION_RESPONSE" />
        <s:enumeration value="DISPOSITION_MESSAGE" />
        <s:enumeration value="DISPOSITION_RESPONSE" />
        <s:enumeration value="TICKET_MESSAGE" />
        <s:enumeration value="TICKET_RESPONSE" />
      </s:restriction>
    </s:simpleType>
    <s:simpleType name="ProcessingModes">
      <s:restriction base="s:string">
        <s:enumeration value="NULL_MODE" />
        <s:enumeration value="NORMAL" />
        <s:enumeration value="TEST_NOTIFY" />
        <s:enumeration value="TEST_PROCESS" />
      </s:restriction>
    </s:simpleType>
```

MessageType. Identifies the document type. The valid values are TICKET_MESSAGE, COLLISION_MESSAGE, DISPOSITION_MESSAGE, TICKET_RESPONSE, COLLISION_RESPONSE, and DISPOSITION_RESPONSE. NULL_TYPE is provided as a default value for when a message

arrives without a value in this element. This situation is then trapped and triggers an error condition. The usage of these values is specified in the section 9.4 "Authorization" of the logical design.

ProcessingMode. Indicates whether the document is a regular document ("NORMAL") or a test document and has to be processed according to a certain rule ("TEST_PROCESS", "TEST_NOTIFY"). Valid values are NORMAL, TEST_PROCESS, and TEST_NOTIFY. NULL_MODE is provided as a default value for when a message arrives without a value in this element. This situation is then trapped and triggers an error condition.

CreationDateTime. Identifies the date and time (in a standard date/time format, specified in ISO 8601) of the message creation by the Sender.

The date and time should include the time zone, and be presented in the following format: YYYY-MM-DDThh:mm:ss.sTZD, where

YYYY	= four-digit year
MM	= two-digit month (01=January, etc.)
DD	= two-digit day of month (01 through 31)
T	= letter "T"
hh	= two digits of hour (00 through 23) (am/pm NOT allowed)
mm	= two digits of minute (00 through 59)
ss	= two digits of second (00 through 59)
s	= one or more digits representing a decimal fraction of a second
TZD	= time zone designator (Z or +hh:mm or -hh:mm)

For example, 1994-11-05T08:15:30-05:00 corresponds to November 5, 1994, 8:15:30 am, US Eastern Standard Time. 1994-11-05T13:15:30Z corresponds to the same instant.

The time zone designator is not mandatory. In case it is omitted, local Pacific Time is assumed.

OriginatorID. JINDEX internal identification of the Participant (original sender of the message). This element is should be left blank when a message is originated by a participant. If a value is present, it will be ignored by JINDEX. Using the x.509 certificate attached to the message, JINDEX will determine the identity of the originator and insert an integer value corresponding to that participant into this element.

OriginatorMessageID. Unique identifier of the message in the system that originated it. For functional responses this element identifies the functional response message and not the message that caused that functional response. For instance, if the message is a TICKET_RESPONSE, the OriginatorMessageID identifies the response and not the message that delivered the traffic ticket. In case of an error when processing the message by the Receiving Web Services JINDEX replies to the Sender with this message ID, as described in 5.1.2.1.11.

PersonalIdentifier. Identifier of the person originated the message. This element is not used in this version and should be an empty string. This element may be used in future versions to identify the individual that generated a particular message.

JindexRecipientID. JINDEX internal identification of the Participant (recipient of the functional response). For Traffic Tickets, Collision Reports, and Disposition Reports this element is not processed and should be empty. The Sender of the functional

response uses the OriginatorID element from the original message to populate the JindexRecipientID element.

JINDEXMessageID. Unique message ID generated in JINDEX and added to JINDEX outbound messages. The non-null element is ignored if present in an inbound message to JINDEX and is overwritten in the outbound message.

7.2.1.3 JINDEX Exchange Response Format

JINDEXAcknowledgement

```
<s:element name="JINDEXAcknowledgment" type="s1:JINDEXAcknowledgment" />
    <s:complexType name="JINDEXAcknowledgment">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="ReturnCode" type="s1:ReturnCodeTypes" />
        <s:element minOccurs="1" maxOccurs="1" name="MessageID" type="s:string" />
      </s:sequence>
    </s:complexType>
    <s:simpleType name="ReturnCodeTypes">
      <s:restriction base="s:string">
        <s:enumeration value="NULL_RETURN" />
        <s:enumeration value="P" />
        <s:enumeration value="E24" />
        <s:enumeration value="E26" />
        <s:enumeration value="E30" />
        <s:enumeration value="E32" />
        <s:enumeration value="F" />
        <s:enumeration value="T" />
      </s:restriction>
    </s:simpleType>
```

ReturnCode. In case of success, Participants will return "P" (for Processing). In case of an error, Participants will return one of error codes described in section 8.1 (E24, E26, E30, E32, F or T).

OriginalMessageID. Identifier of the message generated by the system that originated the message. In this element JINDEX returns the value received in the OriginatorMessageID element. Participants would populate this element with the JINDEXMessageID from messages received from JINDEX.

7.3 JINDEX Web Service WSDL

The following is the JINDEX Exchange API WSDL developed and approved by the SECTOR Technical Advisory Group on September 29th, 2006.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                  xmlns:tns="http://WSDIS.eTrip.ReceivingServices/"
                  xmlns:s1="http://WDIS.E-Trip"
                  xmlns:s="http://www.w3.org/2001/XMLSchema"
                  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  targetNamespace="http://WSDIS.eTrip.ReceivingServices/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">JINDEX Messaging Web
Service</wsdl:documentation>

  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://WSDIS.eTrip.ReceivingServices/">
      <s:import namespace="http://WDIS.E-Trip" />
      <s:element name="JINDEXExchange">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" ref="s1:JINDEXDocument" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="JINDEXExchangeResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" ref="s1:JINDEXAcknowledgment" />
          </s:sequence>
        </s:complexType>
      </s:element>
       </s:schema>
    <s:schema elementFormDefault="qualified" targetNamespace="http://WDIS.E-Trip">
      <s:element name="JINDEXDocument" type="s1:JINDEXDocument" />
      <s:complexType name="JINDEXDocument">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="JINDEXParameter" type="s1:JINDEXParameters" />
          <s:element minOccurs="1" maxOccurs="1" name="JINDEXMessage">
            <s:complexType mixed="true">
              <s:sequence>
                <s:any />
              </s:sequence>
            </s:complexType>
          </s:element>
        </s:sequence>
      </s:complexType>
      <s:complexType name="JINDEXParameters">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="MessageType" type="s1:MessageTypes" />
          <s:element minOccurs="1" maxOccurs="1" name="ProcessingMode" type="s1:ProcessingModes" />
          <s:element minOccurs="1" maxOccurs="1" name="CreateDateTime" type="s:dateTime" />
```

```
        <s:element minOccurs="1" maxOccurs="1" name="OriginatorID" type="s:int" />
        <s:element minOccurs="1" maxOccurs="1" name="OriginatorMessageID" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="PersonalIdentifier" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="JINDEXRecipientID" type="s:int" />
        <s:element minOccurs="0" maxOccurs="1" name="JINDEXMessageID" type="s:int" />
      </s:sequence>
    </s:complexType>
    <s:simpleType name="MessageTypes">
      <s:restriction base="s:string">
        <s:enumeration value="NULL_MESSAGE" />
        <s:enumeration value="COLLISION_MESSAGE" />
        <s:enumeration value="COLLISION_RESPONSE" />
        <s:enumeration value="DISPOSITION_MESSAGE" />
        <s:enumeration value="DISPOSITION_RESPONSE" />
        <s:enumeration value="TICKET_MESSAGE" />
        <s:enumeration value="TICKET_RESPONSE" />
      </s:restriction>
    </s:simpleType>
    <s:simpleType name="ProcessingModes">
      <s:restriction base="s:string">
        <s:enumeration value="NULL_MODE" />
        <s:enumeration value="NORMAL" />
        <s:enumeration value="TEST_NOTIFY" />
        <s:enumeration value="TEST_PROCESS" />
      </s:restriction>
    </s:simpleType>
    <s:element name="JINDEXAcknowledgment" type="s1:JINDEXAcknowledgment" />
    <s:complexType name="JINDEXAcknowledgment">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="ReturnCode" type="s1:ReturnCodeTypes" />
        <s:element minOccurs="1" maxOccurs="1" name="MessageID" type="s:string" />
      </s:sequence>
    </s:complexType>
    <s:simpleType name="ReturnCodeTypes">
      <s:restriction base="s:string">
        <s:enumeration value="NULL_RETURN" />
        <s:enumeration value="P" />
        <s:enumeration value="E24" />
        <s:enumeration value="E26" />
        <s:enumeration value="E30" />
        <s:enumeration value="E32" />
        <s:enumeration value="F" />
        <s:enumeration value="T" />
      </s:restriction>
    </s:simpleType>
  </s:schema>
 </wsdl:types>

 <wsdl:message name="JINDEXExchangeSoapIn">
  <wsdl:part name="parameters" element="tns:JINDEXExchange" />
 </wsdl:message>
 <wsdl:message name="JINDEXExchangeSoapOut">
  <wsdl:part name="parameters" element="tns:JINDEXExchangeResponse" />
 </wsdl:message>
 <wsdl:portType name="JINDEXMessagingServiceSoap">
  <wsdl:operation name="JINDEXExchange">
   <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Listener method for receiving messages from the
JINDEX.</wsdl:documentation>
   <wsdl:input message="tns:JINDEXExchangeSoapIn" />
   <wsdl:output message="tns:JINDEXExchangeSoapOut" />
  </wsdl:operation>
 </wsdl:portType>
```

```
  <wsdl:binding name="JINDEXMessagingServiceSoap" type="tns:JINDEXMessagingServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="JINDEXExchange">
      <soap:operation soapAction="http://WSDIS.eTrip.ReceivingServices/JINDEXExchange" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="JINDEXMessagingServiceSoap12" type="tns:JINDEXMessagingServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="JINDEXExchange">
      <soap12:operation soapAction="http://WSDIS.eTrip.ReceivingServices/JINDEXExchange" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
  </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="JINDEXMessagingService">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">JINDEX Messaging Web
Service</wsdl:documentation>
    <wsdl:port name="JINDEXMessagingServiceSoap" binding="tns:JINDEXMessagingServiceSoap">
      <soap:address
location="https://businessqa.wsdot.wa.gov/highwaysafety/collision/report/electronic/JINDEXMessagingService.asmx" />
    </wsdl:port>
    <wsdl:port name="JINDEXMessagingServiceSoap12" binding="tns:JINDEXMessagingServiceSoap12">
      <soap12:address
location="https://businessqa.wsdot.wa.gov/highwaysafety/collision/report/electronic/JINDEXMessagingService.asmx" />
    </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

8.0 Business Rules Layer and Data Layer

8.1 Message Statuses

A message position in the JINDEX message flow is identified by the message Status field. The last Status information is saved in JINDEX Message Log and may be delivered to Participants via Notification Services.

Message statuses:

1. P (Processing). BizTalk got the message for processing. JINDEX accepts responsibility for the message, and the message Sender is informed by receiving Success return code.
2. E20 (Error). The Message Type is invalid.

   (NOTE: In release 1.10 and higher of the E-Trip Messaging system, Message Type is checked at the Web Service layer as an enumerated value in the WSDL. As such, E20 should never appear as a message status under normal conditions. Because of the critical nature of this element, the BizTalk logic still checks the validity of the value to ensure that the WSDL, and the routing logic stay in sync.)
3. E22 (Error). The Processing Mode is invalid.

   (NOTE: In release 1.10 and higher of the E-Trip Messaging system, Processing Mode is checked at the Web Service layer as an enumerated value in the WSDL. As such, E22 should never appear as a message status under normal conditions. Because of the critical nature of this element, the BizTalk logic still checks the validity of the value to ensure that the WSDL, and the routing logic stay in sync.)
4. E24 (Error). The Sender is not part of JINDEX Participant list.
5. E26 (Error). The Sender is not authorized to send messages of the given type and in the given mode.
6. E30 (Error). The JINDEXRecipientID parameter element is empty when a value was expected.
7. E32 (Error). The JINDEXRecipientID parameter does not match any valid Participant or there is no entry in the Routing table.
8. E34 (Error). The destination URI is not set (or blocked).
9. E50 (Error). Any other error occurred during message processing.
10. S (Ready to be sent). BizTalk completed message processing; the message is ready to be sent.
11. C (Completed). Recipient successfully received the message.
12. F (Failure). Recipient responded with failure.
13. E60 (Error). Recipient responded with anything other than C or F.

14. T (Test-Notify Completed). Message has been processed successfully, but was not sent. Message information is stored in the Message Log.

When the message is successfully delivered or the message can not be processed (as a result of an error), the message is removed from the processing loop. For security reasons JINDEX does not archive the messages. The following information is stored in the system during processing:

- JINDEX Message ID (unique integer identifier generated when the message is accepted by JINDEX)
- JINDEXMessage (XML document created by the Message Originator)
- MessageType (assigned by the Message Originator)
- ProcessingMode (assigned by the Message Originator)
- CreationDateTime (populated by the Sender (either the message originator or JINDEX))
- OriginatorMessageID (assigned by the Message Originator)
- PersonalIdentifier (assigned by the Message Originator, for future versions)
- Sender X.509 Certificate identification (received with the message (either the message originator or JINDEX))
- OriginatorID (if blank field is received, will be filled in later)
- JINDEXRecipientID (assigned by the Sender of a functional response)
- Destination URI (determined from routing table within JINDEX)
- Received Date/Time (the time when JINDEX received the message)
- Message Status
- Modification Date/Time
- Priority (for future versions)

The OriginatorID is filled in when JINDEX identifies the sender using the senders X.509 certificate.

8.2 Archiving Messages in the Message Log

Message Log keeps information about messages processed by JINDEX for auditing purposes. For security reasons the message itself is not archived. When JINDEX removes a message from the Processing Queue (the message could be successfully processed or an error could occur), the following information is stored in the log:

- JINDEXMessageID
- MessageType
- ProcessingMode
- CreationDateTime
- OriginatorMessageID
- PersonalIdentifier (identifies the person, originated the document)

- OriginatorID (unique identifier of the Participant in JINDEX)
- JINDEXRecipientID (assigned by the Sender for functional responses)
- Destination URI (assigned by JINDEX)
- Received Date/Time
- Last Message Status
- Completed Date/Time (the time when JINDEX submitted the message or failed with error)

8.3 Authentication

Authentication of the message senders is performed based on the client X.509 certificates. JINDEX will keep Participant X.509 certificates in the Windows Server Certificate Storage. When a message arrives, the certificate encrypted into the message, will be examined and validated against the stored certificates. Each participant will have a unique participant ID assigned by JINDEX and kept in JINDEX Participant table. Participant IDs will be matched with X.509 certificates based on the combination of the Issuer and Certificate Number fields.

DIS will provide procedures for securely exchanging certificates with participants and will be responsible for maintaining the authentication table.

8.4 Authorization

Before any message is processed and routed through JINDEX, the message Sender has to be authorized. The authorization process has to confirm that the message Sender is authorized to send a message of the specified type to be processed in the specified mode. DIS will provide procedures and will be responsible for maintaining the authorization table.

The following are the available message processing modes for different originating Participants and different message types in the Q/A test environment:

Q/A Test Environment		
Originating Participant	**Message Type**	**Available Message Processing Modes**
{LEA}_SECTOR	TICKET_MESSAGE	Normal; Test- Process; Test-Notify
	COLLISION_MESSAGE	Normal; Test- Process; Test-Notify
	DISPOSITION_MESSAGE	BLOCKED
	TICKET_RESPONSE	Test- Process; Test-Notify
	COLLISION_RESPONSE	Test- Process; Test-Notify
	DISPOSITION_RESPONSE	BLOCKED
AOC	TICKET_MESSAGE	Test- Process; Test-Notify
	COLLISION_MESSAGE	BLOCKED

Q/A Test Environment		
Originating Participant	**Message Type**	**Available Message Processing Modes**
	DISPOSITION_MESSAGE	Normal; Test- Process; Test-Notify
	TICKET_RESPONSE	Normal; Test- Process; Test-Notify
	COLLISION_RESPONSE	BLOCKED
	DISPOSITION_RESPONSE	Test- Process; Test-Notify
DOL	TICKET_MESSAGE	BLOCKED
	COLLISION_MESSAGE	Test- Process; Test-Notify
	DISPOSITION_MESSAGE	Test- Process; Test-Notify
	TICKET_RESPONSE	BLOCKED
	COLLISION_RESPONSE	Test- Process; Test-Notify
	DISPOSITION_RESPONSE	Normal; Test- Process; Test-Notify
WSDOT	TICKET_MESSAGE	BLOCKED
	COLLISION_MESSAGE	Test- Process; Test-Notify
	DISPOSITION_MESSAGE	BLOCKED
	TICKET_RESPONSE	BLOCKED
	COLLISION_RESPONSE	Normal; Test- Process; Test-Notify
	DISPOSITION_RESPONSE	BLOCKED

Table 1 – QA Test Environment Authorization Table

The following are the available message processing modes for different originating Participants and different message types in the production environment:

Production Environment		
Originating Participant	**Message Type**	**Available Message Processing Modes**
{LEA}_SECTOR	TICKET_MESSAGE	Normal; Test- Process; Test-Notify
	COLLISION_MESSAGE	Normal; Test- Process; Test-Notify
	DISPOSITION_MESSAGE	BLOCKED
	TICKET_RESPONSE	BLOCKED
	COLLISION_RESPONSE	BLOCKED
	DISPOSITION_RESPONSE	BLOCKED
AOC	TICKET_MESSAGE	BLOCKED
	COLLISION_MESSAGE	BLOCKED

Production Environment		
Originating Participant	**Message Type**	**Available Message Processing Modes**
	DISPOSITION_MESSAGE	Normal; Test- Process; Test-Notify
	TICKET_RESPONSE	Normal; Test- Process; Test-Notify
	COLLISION_RESPONSE	BLOCKED
	DISPOSITION_RESPONSE	BLOCKED
DOL	TICKET_MESSAGE	BLOCKED
	COLLISION_MESSAGE	BLOCKED
	DISPOSITION_MESSAGE	BLOCKED
	TICKET_RESPONSE	BLOCKED
	COLLISION_RESPONSE	BLOCKED
	DISPOSITION_RESPONSE	Normal; Test-Process; Test-Notify
WSDOT	TICKET_MESSAGE	BLOCKED
	COLLISION_MESSAGE	BLOCKED
	DISPOSITION_MESSAGE	BLOCKED
	TICKET_RESPONSE	BLOCKED
	COLLISION_RESPONSE	Normal; Test-Process; Test-Notify
	DISPOSITION_RESPONSE	BLOCKED

Table 2 – Production Environment Authorization Table

8.5 Routing

8.5.1 Message Types and Processing Modes

Routing is based on Message Types and Processing Modes.

Acceptable **Message Types** are the following:

1. TICKET_MESSAGE
2. TICKET_RESPONSE
3. COLLISION_MESSAGE
4. COLLISION_RESPONSE
5. DISPOSITION_MESSAGE
6. DISPOSITION_RESPONSE

Acceptable **Processing Modes** are the following:

1. Normal
2. Test-Process
3. Test-Notify

8.5.2 Routing Types

1. All message types except Traffic Ticket Functional Response and Collision Functional Response types sent in the Normal or Test-Process modes are delivered to the predetermined Recipients. The routing algorithm is based on the following table:

Normal and Test-Process Processing Modes		
Originating Participant	**Message Type**	**Recipient**
{LEA}_SECTOR	TICKET_MESSAGE	AOC
	COLLISION_MESSAGE	WSDOT, DOL
AOC	DISPOSITION_MESSAGE	DOL
	TICKET_RESPONSE	Originating LEA
DOL	DISPOSITION_RESPONSE	AOC
WSDOT	COLLISION_RESPONSE	Originating LEA & DOL

Table 3 – Routing Table

2. A functional response to a Traffic Ticket or Collision message has to be delivered to a particular Law Enforcement Agency that originated the original message. When JINDEX passes the original message to a Participant, it submits the ORIGINATORID as one of the parameters. This ID has to be returned by the Participant, sending the response as the JINDEXRecipientID parameter. JINDEX uses this parameter to identify the ultimate recipient of the functional response.
3. The third routing type applies to all messages being sent in the Test-Notify mode. After being processed, the messages are logged in the Message Log for delivery by Notification Services.

8.6 Addressing

8.6.1 Addressing Information

JINDEX determines the destination URI for messages to be delivered based on the Recipient ID, current status of the Recipient, Message Type, and Processing Mode.

Participants may use different URIs for different message types. The status of the Recipient is preset (except Not Responding) and can be changed by JINDEX administrator.

The following table describes the parameters to be considered to determine the destination address (URI).

Column	Description	Values
Recipient	Participant where the messages will be sent.	WSP; Seattle PD DOL; AOC; WSDOT
Recipient Status	Is the Participant accepting these messages and in what processing mode: • **NORMAL** – Participant is in production mode for this message type. Messages sent with a processing flag of Normal will be passed to the "Normal URI". • **TEST** – Participant is not in production mode for this message type. Messages received with a "Normal" processing flag will be treated as "Hold". Messages flagged as "Test-Process" will be sent to the Test Address. • **HOLD** – Participant is temporarily not accepting messages. Messages will be addressed to their proper destination, but will be queued. • **DRBC** – Participant has executed its Disaster Recovery / Business Continuity (DRBC) plan. Normal messages will be addressed to the DRBC address. Test-Process messages will be processed normally, but will be queued. • **NOT RESPONDING** – communication with the Participant failed. Messages are queued for resending.	Normal, Test, Hold, DRBC, Not Responding
Normal Address	Address where messages should be sent whose process flag is set to "Normal" and Participant Status is set to Normal.	
Test Address	Address where messages should be sent whose process flag is set to "Test-Process", and where the Participant Status is set to Normal or Test.	
DRBC Address	Address where messages should be sent whose process flag is set to "Normal" and where the Participant status is set to "DRBC".	

Table 4 – Addressing rules

8.7 Error Handling

In case of a processing error the following actions will be taken:

1. Message information will be logged in the Message Log if an error occurred in BizTalk. It will specify error code, additional error information, and other message parameters.
2. If SQL Server is not responding or in case of another critical system error, JINDEX attempts to log the message in the Windows Application log.
3. MS SQL Server Notification Services (SSNS) will pick this record from the Message Log and send an e-mail notification to an appropriate system administrator.

8.7.1 Message Log

Message Log holds information about messages along with the error code.

8.7.2 Error Categories

1. Message validation errors.
2. Sender validation errors.
3. Message schema validation errors.
4. Communication errors.
5. Other party errors.
6. JINDEX internal errors.

Error categories will be used to set up subscriptions for notifications.

8.8 Notification

Notifications are implemented by means of MS SQL Service Notification Services (SSNS). The notification procedure will perform periodic checking of specified SQL Sever tables based on given queries, and send completely configurable notifications to notification subscribers by means of different communication tools (e-mail primarily). To submit the notifications via e-mail an SMTP server has to be set up. However, notifications can be sent to a variety of devices: cellular phones, PDAs, or MSN messenger accounts.

8.8.1 Notification Message Structure

1. Notification sender: JINDEX. The From SMTP field is the same for all messages, for example, jindex@jindex.com.
2. Notification recipient(s): configurable address(es) based on the original message type and configuration message type.
3. Notification body.

For security reasons e-mail notifications will not have any original message attached. The following information can be sent in the notification:

- JINDEX Unique Message ID
- Message Type
- Processing Mode
- Creation Date/Time
- Original Message ID
- Personal Identifier
- Sender ID
- Recipient ID
- Destination URI
- Received Date/Time
- Last Message Status
- Last Modification Date/Time
- Notification Description (error information, in case if an error occurs)

8.8.2 Notification Subscriptions

Notifications are generated by JINDEX and sent to notification subscribers. Each subscription will be configured based on

1. Original message type;
2. Processing mode;
3. Sender ID;
4. Message category (such as an error category, for example).

8.9 Auditing

Auditing will be performed periodically based on JINDEX Message Logs. Parameterized queries can be run against the log with the parameters corresponding to the fields in the log (see 9.2).